**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

APPELLANT:	Katrin Reisinger	CONFIRMATION NO. 5666
SERIAL NO.:	10/797,838	GROUP ART UNIT: 3628
FILED:	March 10, 2004	EXAMINER: Eric Liou
TITLE:	APPARATUS FOR AUTOMATIC DETERMINATION OF A PRODUCT DESCRIPTION FOR DISPLAY BY MEANS OF A MAIL-PROCESSING DEVICE	

MAIL STOP APPEAL BRIEF-PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

APPELLANT'S APPEAL BRIEF

S I R:

In accordance with the provisions of 37 C.F.R. §41.37, Appellant herewith submits her main brief in support of the appeal of the above-referenced application.

TABLE OF CONTENTS

TABLE OF AUTHORITIES

REAL PARTY IN INTEREST:

The real party in interest is the assignee of the present application, which is named on the Assignment as Francotyp-Postalia AG & CO KG, now d/b/a Francotyp-Postalia GmbH, a German corporation.

RELATED APPEALS AND INTERFERENCES:

There are no related appeals and no related interferences.

STATUS OF CLAIMS:

Claims 1-4 are the subject of the present appeal, and constitute all pending claims of the application. No claim was added or canceled during prosecution of this application.

STATUS OF AMENDMENTS:

No Amendment was filed subsequent to the Final Rejection.

SUMMARY OF CLAIMED SUBJECT MATTER:

Claim 1, which is the only independent claim on appeal, is set forth below with exemplary citations to the present disclosure for relevant claim elements.

1. A mail-processing device comprising:

a microprocessor (µP7, Fig. 1);

a keyboard with operating elements connected to said microprocessor for entering shipping information into said microprocessor (keyboard 6, Fig. 1);

a working memory accessible by said microprocessor containing mail-item-related data values (RAM5, Fig.1);

a programmable memory and a program memory accessible by said microprocessor (EEPROM2, Fig. 1);

in at least one of said program memory and said programmable memory(p.9, l.7-9), a first memory area containing a program that evaluates said mail-item-related data values stored in the working memory to cause said mail-item-related data values to be permanently or temporarily stored (memory range D; p.9, l.10-12), a second memory area containing a first table for indices respectively assigned to different country-specific, postal authority-defined product codes (memory range E; p.9, l.13-14), said product codes being ascendingly or consecutively stored in said table in a column and said table having a second column, in parallel with said first column, containing indices for different product descriptions (p.9, l.15-16), and a third memory range for storage of a further table for said product descriptions respectively assigned to said indices in said second column (memory range F; p.l.17-19); and

said microprocessor being programmed by said program for evaluating the mail-item-related data values stored in the working memory by accessing said table containing said first and second columns to automatically determine a product code and a product description for said service product, and to supply as an output a text for said product description for generating a printout thereof (p.9, l. 20-25).

A copy of Figure 1 of the application as originally filed is submitted herewith as Exhibit "A".

GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL:

The sole issue to be reviewed on appeal is whether the subject matter of claims 1-4 would have been obvious to a person of ordinary skill in the field of designing mail processing devices, under the provisions of 35 U.S.C. §103(a), based on the teachings of United States Patent No. 5,040,132 (Schuricht et al., Exhibit "B") and United States Patent No. 5,602,382 (Ulvr et al., Exhibit "C").

ARGUMENT:

Rejection of Claims 1-4 Under 35 U.S.C. §103(a) Based on Schuricht et al. and Ulvr et al.

The subject matter of the claims on appeal is a mail-processing device that allows a product code to be automatically retrieved from a memory, upon the entry of shipping information into the mail-processing device and to supply, as an output, text for the product description for generating a printout thereof.

The term "product code" is a term with a specific, well-documented meaning in the context of mail processing. As explained in the first full paragraph at page 3 of the present specification, a product code is a specific definition pertaining to a specific mailing category that is defined by the governmental postal authorities in some countries, such as in Germany and in Canada. The product code designates additional services, beyond basic mailing, that are requested by the mailer, such as overnight delivery, registered mail, etc. The product code must be included in the franking imprint according to the postal regulations in these countries, but this code is simply a number and therefore does not, by itself, provide explanatory information to a user who has not memorized all of the relevant product codes. As explained at page 3 of the present specification, this therefore necessitates extra steps by the user in generating the franking imprint.

Appellant also submitted documentary evidence during prosecution to substantiate this point. Since this does not appear to be disputed by the Examiner, inclusion of that documentary evidence as an exhibit hereto is not necessary.

In accordance with the present invention, all of the available services from the postal authority are displayed to the user in a menu. The user then selects the desired service from the menu. It still remains, however, to identify the postal authority-defined product code for the selective service. Conventionally, this would have required the user to then peruse another menu showing the product codes, and the user would then have to select the appropriate product code. In accordance with the present invention, this extra step on the part of the user is eliminated by automatically identifying and selecting the appropriate product code, from a stored table, simply upon the initial entry by the user of the appropriate shipping information.

Such an apparatus is not disclosed or suggested in the Schuricht et al. reference. This is for several reasons.

The patent that issued as the Schuricht et al. reference was originally filed on May 15, 1989, which was long before any type of governmentally-defined product codes were in existence. Therefore, the Schuricht et al. reference does not and could not concern such governmentally-defined product codes. Moreover, the Schuricht et al. reference is assigned to a United States company, and as noted above, even as of today such product codes are not required in the United States. Therefore, there was no reason for Schuricht et al. to include anything in the device disclosed in the Schuricht et al. reference that would facilitate the identification of a governmentally-defined product code that is associated with a selected level of service.

This is also made clear in the text of the governmentally-defined reference itself, at column 2, lines 37-39 and 45-49, which explicitly require manual entries in order to retrieve the stored information. It is true that the Schuricht et al. reference discloses *storage* of certain types of codes associated with certain types of services but, for the reasons noted above, these are not and cannot be governmentally-defined product codes. Even if the Examiner considers the stored alphanumeric information in Schuricht et al. to conform to some type of generic product code, there is no automatic retrieval of that product code for inclusion in the franking imprint disclosed in the Schuricht et al. reference. Retrieval of those codes in Schuricht et al. still requires an additional manual entry, which is avoided in accordance with the present invention.

In the apparatus disclosed and claimed in the present application, the determination of the product code proceeds automatically in the "background" after the user has entered the appropriate shipping information. This is described in the present specification at page 4, line 17 through page 5, line 5.

The Ulvr et al reference, like the Schuricht et al reference, has a filing date and a publication date that are significantly earlier than the relatively recent conception and usage of product codes, and therefore the Ulvr et al references does not and cannot provide any guidance to a person of ordinary skill in the field of designing postage meters with regard to the incorporation of an automatic designation of a product code in such a postage meter.

The Examiner stated the Ulvr et al reference teaches country-specific codes but, as the Examiner has noted, the country-specific codes disclosed in the Ulvr et al reference are country-codes, and are not (and could not be) product codes. The Examiner has unjustifiably, and without any explanation, equated the usage of country codes with the much more recent usage of product codes. These are two completely different items and the use of a country code in no manner guides or otherwise informs a person of ordinary skill in the field of designing postage meters as to how automatic implementation of product codes can be done.

A country-code is used for mailing items from countries such as Canada, wherein the postal rate is different dependent on different country destinations for the mailed item. In Canada, for example, if it is desired to mail a letter by surface mail from Canada to the United Kingdom, the user of a postage meter in Canada enters the designation “GB” (standing for Great Britain) in order to access the rate tables for mailing items from Canada to the United Kingdom. Dependent on the weight and size of the item, these postage rate tables that are country-specific for the United Kingdom determine the cost of making a regular surface mailing from Canada to the United Kingdom.

If, however, the person mailing such an item wishes to mail the item with a particular type of service, such as by registered mail, then such a person must undertake an additional step to determine the surcharge or additional fee that is associated with mailing an item by registered mail. This is completely different from the aforementioned determination of the postage fee that occurs due to the item being destined for the United Kingdom.

As noted above, many countries are now requiring that the franking imprint that is printed on the mailed item include a designation of the product code for each type of special service, such as registered mail. Assuming that the user of a postage machine has not memorized each and every product code for each and every different type of product (mailing service), this means that the user of a postage machine must manually consult some type of table in order to know which product code to enter into the postage meter so that the correct designation will be printed in the franking imprint.

The product code concept, and the requirement in some countries (but still not in the United States) of embodying a designation of the product code in the printed franking imprint, have only recently been developed, and such concept and requirements did not even exist at the time contemporaneous with both the Schuricht et al and Ulvr et al references. Therefore, nothing in those references can provide any guidance to a person of ordinary skill in the field of designing postage meters in order to incorporate these very recently-developed concepts and requirements. Since a product code is a designation that is completely separate, and different from, the country code that is disclosed in the Ulvr et al reference, and in fact is used in addition to such a country code, the Ulvr et al reference can provide no guidance that goes beyond the automated entry of country codes. The only location where automated product code entry is disclosed is Appellant's specification. Appellant submits it is the epitome of hindsight and unjustified use of Appellant's disclosure to reject the claims of the present application using references that, because of their dates, do not and cannot provide any insight into the use of product codes in any form, much less in an automated manner.

Moreover, for the reasons discussed above, Appellant submits that there is no basis for the Examiner, or a person of ordinary skill in the relevant technology, to equate country codes with produce codes. If the Examiner, however, is in fact taking this position, then it is incumbent on the Examiner to provide some evidentiary basis for this alleged equality between these two different and separately used items. Despite Appellant having continuously made the above arguments during prosecution, the Examiner has never provided any such evidence in support, and continues to erroneously confuse and equate country codes with product codes. Appellant therefore respectfully submits that the current rejection is not even a prima facie obviousness rejection, since it lacks the rigorous evidentiary support that is necessary to support such a rejection under the provisions of 35 U.S.C. §103(a).

The Federal Circuit stated in *In re Lee* 227 F.3d 1338, 61 U.S.P.Q. 2d 1430 (Fed. Cir. 2002):

> "The factual inquiry whether to combine references must be thorough and searching. ...It must be based on objective evidence of record. This precedent has been reinforced in myriad decisions, and cannot be dispensed with."

Similarly, quoting *C.R. Bard, Inc. v. M3 Systems, Inc.,* 157 F.3d 1340, 1352, 48 U.S.P.Q. 2d 1225, 1232 (Fed. Cir. 1998), the Federal Circuit in *Brown & Williamson Tobacco Court v. Philip Morris, Inc.,* 229 F.3d 1120, 1124-1125, 56 U.S.P.Q. 2d 1456, 1459 (Fed. Cir. 2000) stated:

> [A] showing of a suggestion, teaching or motivation to combine the prior art references is an 'essential component of an obviousness holding'.

In *In re Dembiczak,* 175 F.3d 994,999, 50 U.S.P.Q. 2d 1614, 1617 (Fed. Cir. 1999) the Federal Circuit stated:

> Our case law makes clear that the best defense against the subtle but powerful attraction of a hindsight-based obviousness analysis is rigorous application of the requirement for a showing of the teaching or motivation to combine prior art references.

Consistently, in *In re Rouffet,* 149 F.3d 1350, 1359, 47 U.S.P.Q. 2d 1453, 1459 (Fed. Cir. 1998), the Federal Circuit stated:

> [E]ven when the level of skill in the art is high, the Board must identify specifically the principle, known to one of ordinary skill in the art, that suggests the claimed combination. In other words, the Board must explain the reasons one of ordinary skill in the art would have been motivated to select the references and to combine them to render the claimed invention obvious.

In *Winner International Royalty Corp. v. Wang,* 200 F.3d 1340, 1348-1349, 53 U.S.P.Q. 2d 1580, 1586 (Fed. Cir. 2000), the Federal Circuit stated:

> Although a reference need not expressly teach that the disclosure contained therein should be combined with another, ... the showing of combinability, in whatever form, must nevertheless be clear and particular.

Lastly, in *Crown Operations International, Ltd. v. Solutia, Inc.,* 289 F.3d 1367, 1376, 62 U.S.P.Q. 2d 1917 (Fed. Cir. 2002), the Federal Circuit stated:

> There must be a teaching or suggestion within the prior art, within the nature of the problem to be solved, or within the general knowledge of a person of ordinary skill in the field of the invention, to look to particular sources, to select particular elements, and to combine them as combined by the inventor.

Appellants submit that the decision of the United States Supreme Court in *KSR International Co. v. Teleflex Inc.,* _____U.S. _____, 127 S.Ct. 1727, 82 USPQ 2d 1385 (2007), and the United States Patent and Trademark Office guidelines for applying that decision, support the position of the Appellants. That decision, although stating that it is not always required to point to a specific teaching in a prior art reference in order to substantiate a rejection under 35 U.S.C. §103(a), by no means approved ignoring the above long-standing precedent, and certainly did not represent a blanket overruling of that precedent. In the *KSR* decision, the Supreme Court stated, *under certain circumstances*, it may not be necessary to point to a specific passage in a prior art reference as evidence of motivation, guidance or inducement in order to modify that reference in a manner that obviates the patent claim in question. The Supreme Court stated that if a person of ordinary skill in the art can implement a *predictable variation* and would see the benefit of doing so, Section 103(a) likely bars patentability.

Nevertheless, the Supreme Court also stated that the requirement to find a teaching, suggestion or motivation in the prior art "captures a helpful insight." The Supreme Court stated that although common sense directs caution as to a patent application claiming as innovation the combination of two known devices according to their established functions, it can be important to identify a reason that would have prompted a person of ordinary skill in the art to combine the elements as the new invention does. The Supreme Court, however, stated that not every application requires such detailed reasoning. The Supreme Court stated that helpful insights need not become rigid and mandatory formulas. The Supreme Court only stated that if the requirement to find a teaching, suggestion or motivation is required in such a rigid, formulaic manner, it is then inconsistent with the precedence of the Supreme Court. In fact, the Supreme Court stated that since the "teaching, suggestion or motivation" test was devised, the Federal Circuit doubtless has applied it in accord with these principles in many cases. The Supreme Court stated there is no necessary inconsistency between this test and an analysis conducted under the standards of *Graham v. Deere*. The Supreme Court stated the only error is transforming this general principle into a "rigid rule limiting the obviousness inquiry."

Therefore, Appellants submit this decision of the Supreme Court does not in any manner approve, much less require, the absence of a rigorous evidentiary investigation on the part of the Examiner in order to substantiate most rejections under 35 U.S.C. §103(a). Only under the somewhat unusual, and very limited, circumstances outlined by the Supreme Court in the *KSR* decision might the Supreme Court excuse the absence of such a rigorous evidentiary investigation in reaching a conclusion of obviousness under 35 U.S.C. §103(a).

This view of the *KSR* decision has been substantiated by the United States Court of Appeals for the Federal Circuit in *Takeda Chemical Industries Limited v. Alphapharm Pty.Ltd.*,492 F.3d 1350, 83 U.S.P.Q.2d, 169 (Fed. Cir. 2007), which was one of the earliest decisions of the Federal Circuit after the *KSR* decision was decided by the Supreme Court. The *Takeda* decision concerned a chemical patent that was the subject of an infringement lawsuit, and which was attacked by the infringer on the basis of the claimed subject matter being "obvious to try." After acknowledging that the *KSR* decision held that the teaching-suggestion-motivation test should not be applied rigidly, the Federal Circuit stated that the *KSR* decision actually recognized the value of that test in determining whether the prior art provided a *reason* for one of skill in the art to make the claimed combination. The Federal Circuit stated this is consistent with the Federal Circuit precedent in *In re Dillon,* 919 F.2d 688 (Fed. Cir. 1990) and in *In re Deuel*, 51 F.3d 1552 (Fed. Cir. 1995). The Federal Circuit stated that in cases involving new chemical compounds, it remains necessary to identify some reason that would have led a chemist to modify a known compound in a particular manner to establish *prima facie* obviousness of the new claimed compound. In the *Takeda* decision, the Federal Circuit stated:

> The *KSR* Court recognized that "[w]hen there is a design need or market pressure to solve a problem and there are a finite number of identified, predictable solutions, a person of ordinary skill has good reason to pursue the known options within his or her technical grasp," *KSR*, 127 S.Ct. at 1732. In such circumstances, "the fact that a combination was obvious to try might show that it would obvious under §103." *id.* that is not the case here. Rather than identify predictable solutions for antidiabetic treatment, the prior art disclosed a broad selection of compounds, any one of which could have been selected as a lead compound for further investigation.

In the Office Action dated July 17, 2007, in paragraph 3 at page 2, the Examiner stated that the Appellant's admissions and the aforementioned submitted documentary support relating to product codes were being relied upon by the Examiner to substantiate that the use of product codes was well known before the time of the instant application. Appellant has never denied this fact, and indeed it was the Appellant that called this fact to the attention of the Examiner. If the fact that such product codes were well known before the time of the instant application is considered by the Examiner to be a relevant fact, however, it seems that the Examiner should also find relevant that the use of product codes were not known at the time of the Schuricht et al and Ulvr et al inventions. The Examiner cannot properly rely on the consequences of a particular fact exclusively in a manner that is detrimental to the present Appellant's position, while ignoring other consequences of the same fact that are in support of Appellant's position.

Claims 2-4 add further structure to the non-obvious combination of claim 1, and therefore none of claims 2-4 would have been obvious to a person of ordinary skill in the field of designing mail processing devices under the provisions of 35 U.S.C. §103(a), based on the teachings of Schuricht et al. and Ulvr et al., for the same reasons discussed above in connection with claim 1.

CONCLUSION:

For the foregoing reasons, Appellant respectfully submits the Examiner is in error in law and in fact in rejecting claims 1-4 under 35 U.S.C. §103(a) based on the teachings of Schuricht et al. and Ulvr et al.. Reversal of that rejection is proper, and the same is respectfully requested.

This Appeal Brief is accompanied by electronic payment for the requisite fee in the amount of $510.00.

This Appeal Brief is also accompanied by a Request for One Month Extension of Time, together with electronic payment for the requisite extension fee.

Submitted by,

Steven H. Noll (Reg. 28,982)

SCHIFF, HARDIN LLP
CUSTOMER NO. 26574
Patent Department
6600 Sears Tower
233 South Wacker Drive
Chicago, Illinois 60606
Telephone: 312/258-5790
Attorneys for Appellant.

CLAIMS APPENDIX

1. A mail-processing device comprising:

a microprocessor;

a keyboard with operating elements connected to said microprocessor for entering shipping information into said microprocessor;

a working memory accessible by said microprocessor containing mail-item-related data values;

a programmable memory and a program memory accessible by said microprocessor;

in at least one of said program memory and said programmable memory, a first memory area containing a program that evaluates said mail-item-related data values stored in the working memory to cause said mail-item-related data values to be permanently or temporarily stored, a second memory area containing a first table for indices respectively assigned to different country-specific, postal authority-defined product codes, said product codes being ascendingly or consecutively stored in said table in a column and said table having a second column, in parallel with said first column, containing indices for different product descriptions, and a third memory range for storage of a further table for said product descriptions respectively assigned to said indices in said second column; and said microprocessor being programmed by said program for evaluating the mail-item-related data values stored in the working memory by accessing said table containing said first and second columns to automatically determine a product code and a product description for said service product, and to supply as an output a text for said product description for generating a printout thereof.

2. A mail-processing device as claimed in claim 1 wherein said microprocessor is programmed by said program to determine the index for the product code and to locate the index in said further table for said product description.

3. A mail-processing device as claimed in claim 1 comprising, in at least one of said program memory and said programmable memory, a fourth memory area for storage of an additional further table containing text strings assigned to the respective indices, and wherein said microprocessor is programmed by said program to determine a text string from said additional further table dependent on shipping parameters for said service product and to supply said text strings in said output.

4. A mail-processing device as claimed in claim 1 wherein said program memory is a permanent memory and wherein said programmable memory is a semi-permanent memory.

EVIDENCE APPENDIX

Exhibit A:	Figure 1 of the application as originally filed.
Exhibit B:	United States Patent No. 5,040,132 (Schuricht et al.)
Exhibit C:	United States Patent No. 5,602,382 (Ulvr et al.)

RELATED PROCEEDINGS APPENDIX

None.

CH1\5963535.1

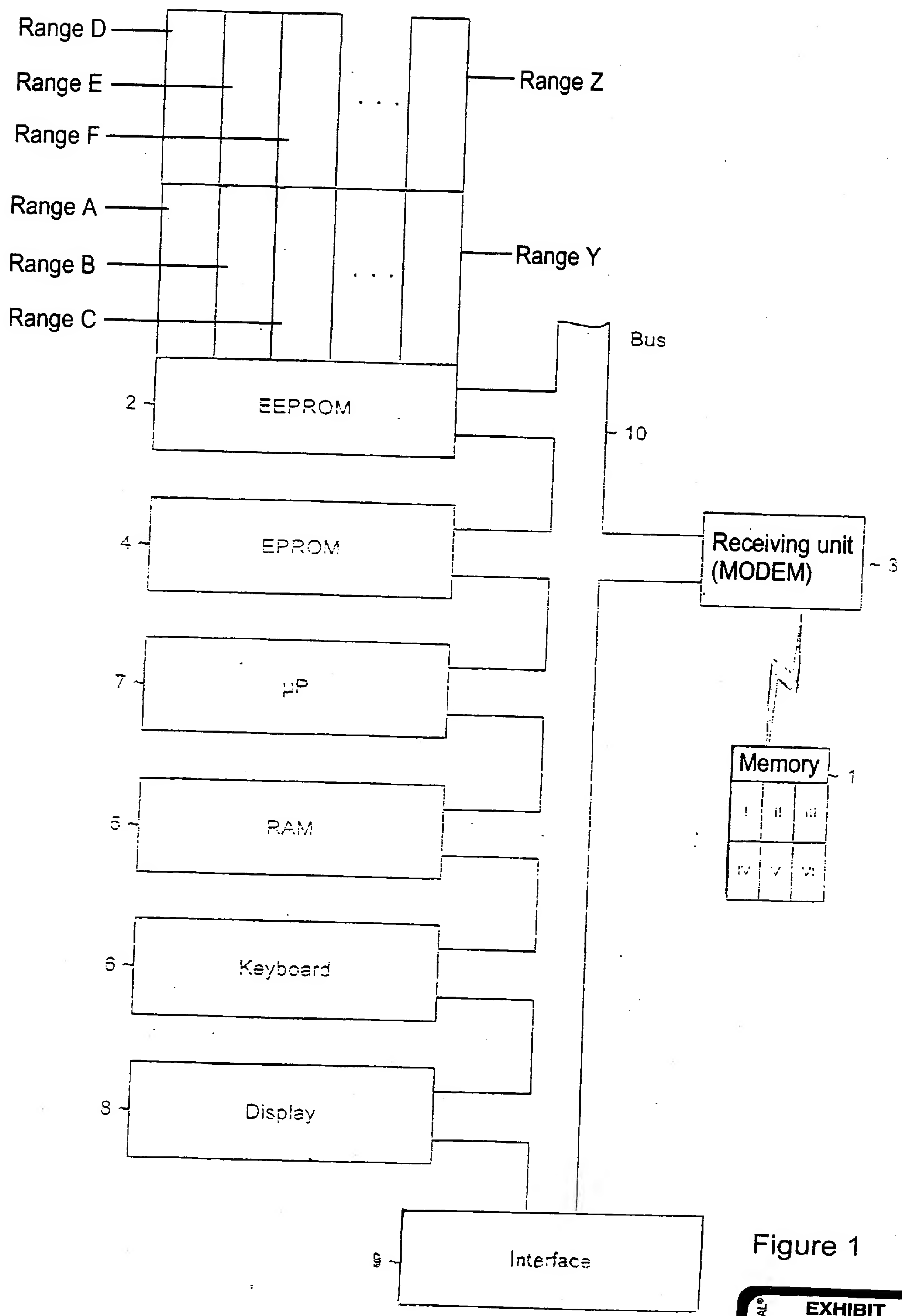


Figure 1

ALL-STATE LEGAL®
EXHIBIT
A

United States Patent [19]
Schuricht et al.

[11] Patent Number: 5,040,132
[45] Date of Patent: Aug. 13, 1991

[54] SYSTEM FOR PREPARING SHIPPING DOCUMENTS

[75] Inventors: Jurgen Schuricht, Lautertal; Reinhold Grocholl, Reichelsheim; Armin Zeiss, Lindenfels/Odw., all of Fed. Rep. of Germany

[73] Assignee: Pitney Bowes Inc., Stamford, Conn.

[21] Appl. No.: 546,656

[22] Filed: Jun. 28, 1990

Related U.S. Application Data

[63] Continuation of Ser. No. 323,760, Mar. 15, 1989, abandoned.

[51] Int. Cl.^{5} G06F 15/20; G06F 15/22

[52] U.S. Cl. 364/523; 364/464.01; 364/464.03; 364/466

[58] Field of Search 364/401, 403, 406, 407, 364/408, 464.01, 464.02, 464.03, 464.04, 466, 523

[56] References Cited

U.S. PATENT DOCUMENTS

4,410,961	10/1983	Dlugos et al.	364/464.03 X
4,420,819	12/1983	Price et al.	364/466 X
4,430,716	2/1984	Dlugos et al.	364/464.03 X
4,481,587	11/1984	Daniels, Jr.	364/466
4,574,352	3/1986	Coppola et al.	364/466
4,713,761	12/1987	Sharpe et al.	364/406
4,797,832	1/1989	Axelrod et al.	364/464.02 X
4,839,813	6/1989	Hills et al.	364/464.03
4,941,091	7/1990	Breault et al.	364/406
4,956,782	9/1990	Freeman et al.	364/464.03

OTHER PUBLICATIONS

Computerized Parcel Shipping System Summary, Pitney Bowes, 1986, pp. 1-45.
"The Mailroom Can Help Your Business Prosper", by Robert C. Lawrence, Jr., The Office, Aug. 1984, pp. 79-80.
"Helping to Move Freight Forward", Computer Systems, Oct. 1985, pp. 19-20.

Primary Examiner—Dale M. Shaw
Assistant Examiner—Michael A. Jaffe
Attorney, Agent, or Firm—Nathaniel Levin; Melvin J. Scolnick; Robert H. Whisker

[57] ABSTRACT

A system for preparing shipping documents includes a processor, a keyboard, a display, a memory and a printer. Associated with the printer are a single sheet feeder, a batch feeder and an endless form feeder. The system is capable of preparing a wide variety of waybills and other shipping forms. The system is also able to prepare automatically all shipping documents required for a mass shipment to a user-defined group of recipients. A user activated "help" function provides information specific to the type of data that the user is attempting to enter at the time the help function is activated. There is also a "rare shopping" function that allows the user to define for each shipping mode a group of other shipping modes for rate comparison purposes.

20 Claims, 19 Drawing Sheets

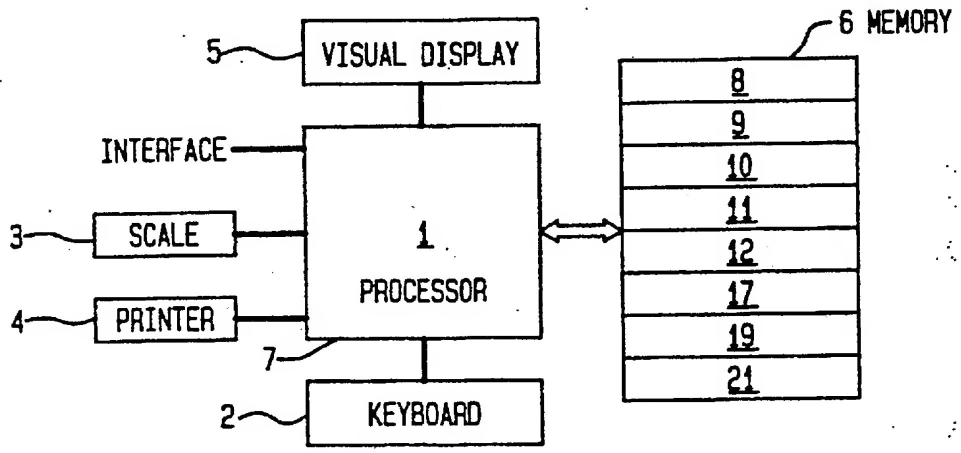


FIG. 1

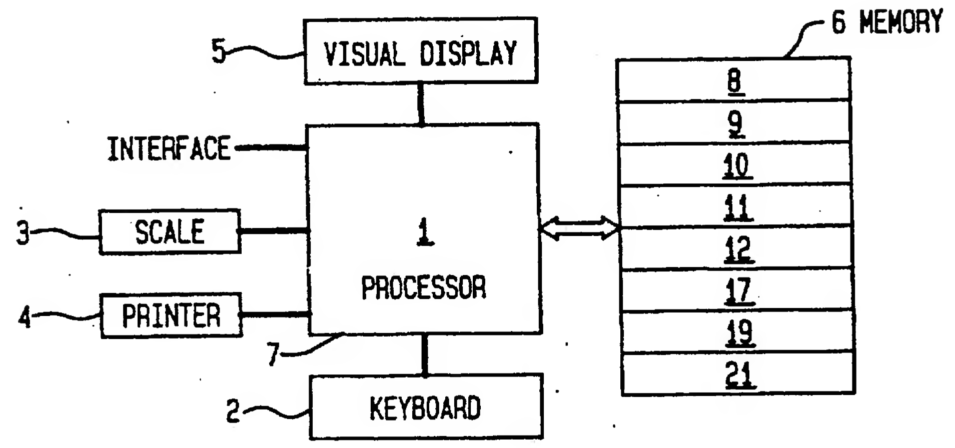


FIG. 2A

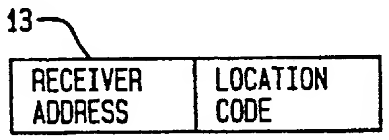


FIG. 2B

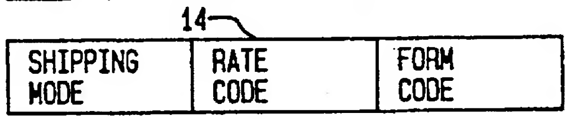


FIG. 2C

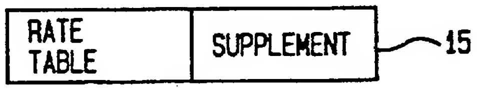


FIG. 2D

PRINT FORM
16

FIG. 2E

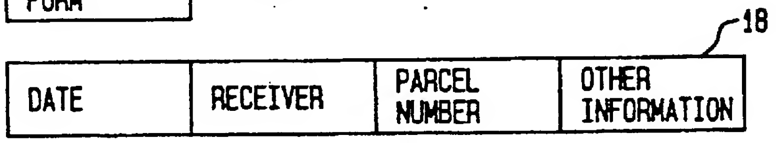


FIG. 2F

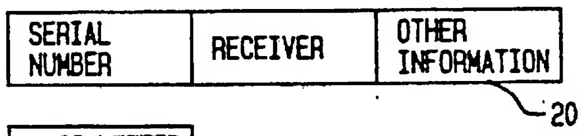


FIG. 2G

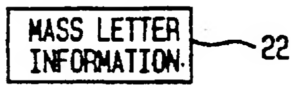
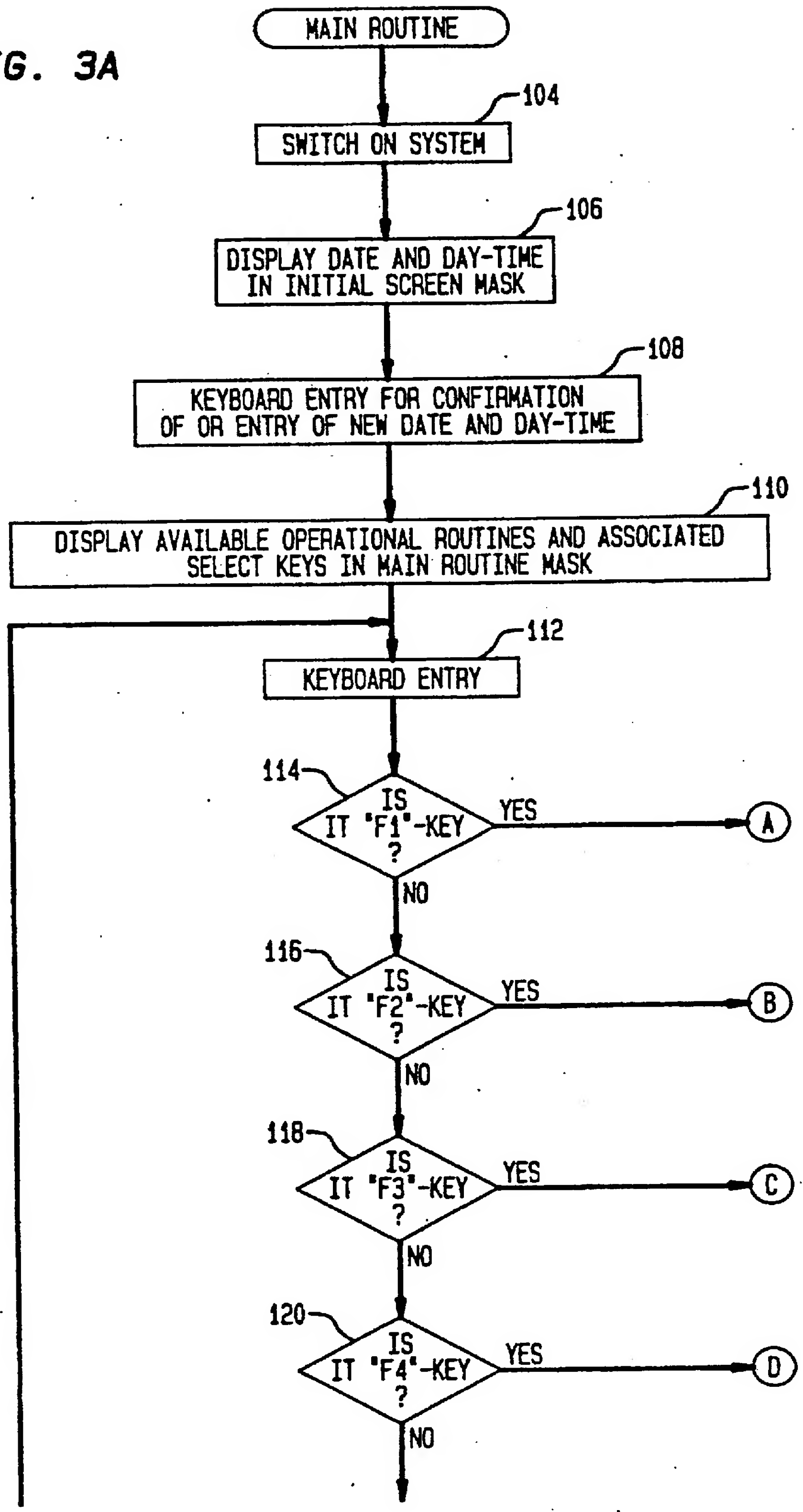


FIG. 3A


MAIN ROUTINE
104
SWITCH ON SYSTEM
106
DISPLAY DATE AND DAY-TIME IN INITIAL SCREEN MASK
108
KEYBOARD ENTRY FOR CONFIRMATION OF OR ENTRY OF NEW DATE AND DAY-TIME
110
DISPLAY AVAILABLE OPERATIONAL ROUTINES AND ASSOCIATED SELECT KEYS IN MAIN ROUTINE MASK
112
KEYBOARD ENTRY
114
IS IT "F1"-KEY ?
YES
A
NO
116
IS IT "F2"-KEY ?
YES
B
NO
118
IS IT "F3"-KEY ?
YES
C
NO
120
IS IT "F4"-KEY ?
YES
D
NO

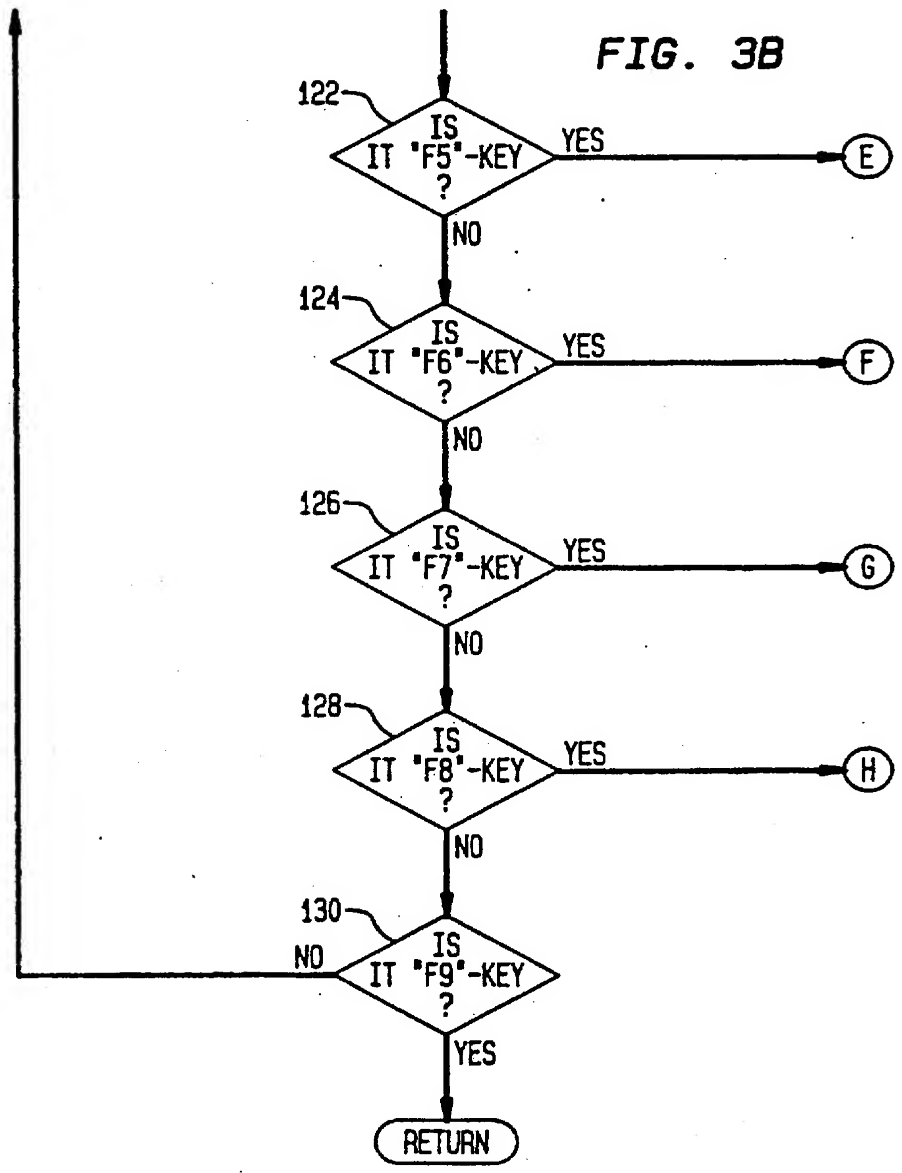

122
IS IT "F5"-KEY ?
YES
E
NO
124
IS IT "F6"-KEY ?
YES
F
NO
126
IS IT "F7"-KEY ?
YES
G
NO
128
IS IT "F8"-KEY ?
YES
H
NO
130
IS IT "F9"-KEY ?
NO
YES
RETURN

FIG. 3B

FIG. 4A

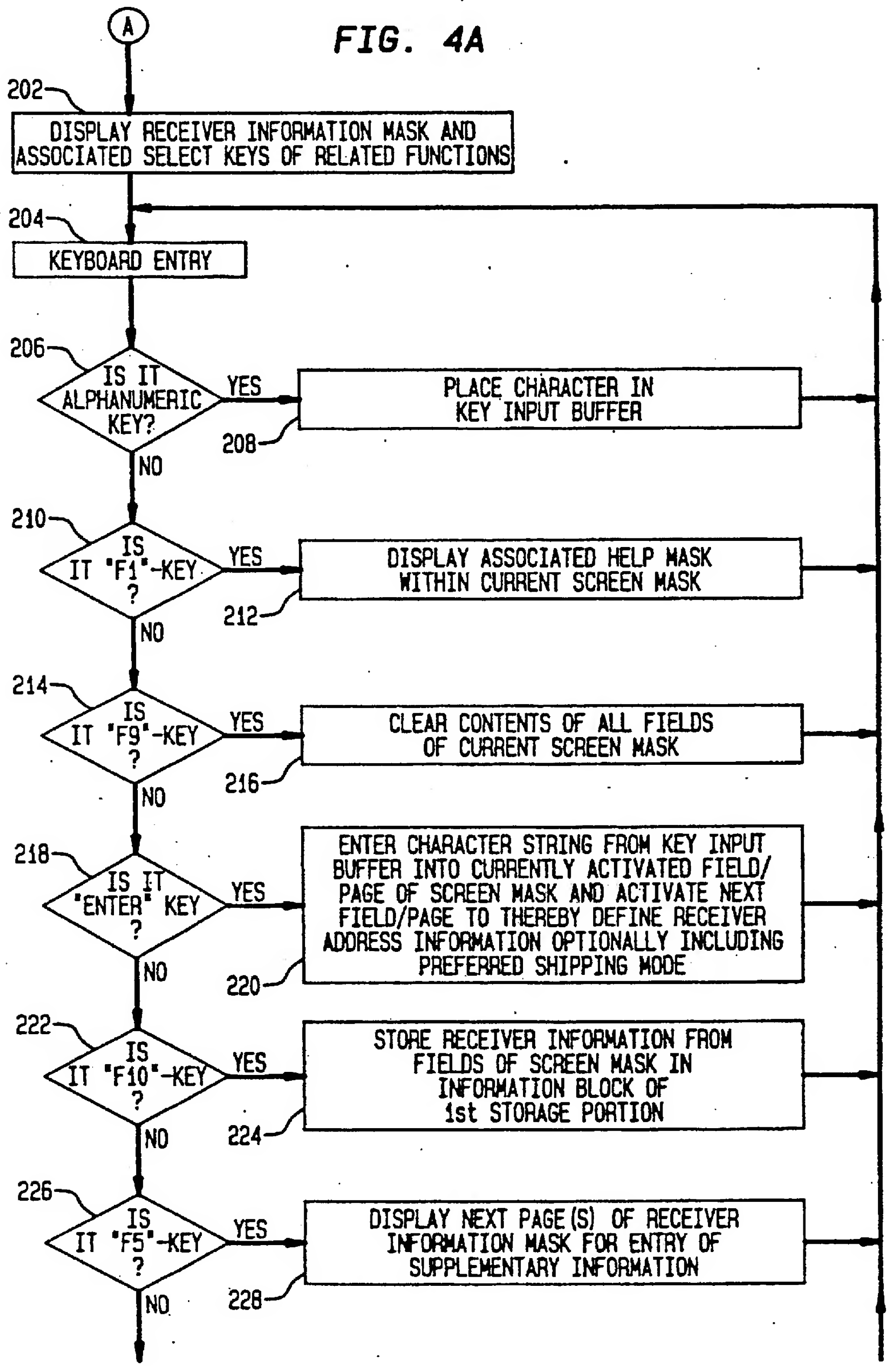

A
202
DISPLAY RECEIVER INFORMATION MASK AND ASSOCIATED SELECT KEYS OF RELATED FUNCTIONS
204
KEYBOARD ENTRY
206
IS IT ALPHANUMERIC KEY?
YES
NO
PLACE CHARACTER IN KEY INPUT BUFFER
208
210
IS IT "F1"-KEY ?
YES
NO
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
212
214
IS IT "F9"-KEY ?
YES
NO
CLEAR CONTENTS OF ALL FIELDS OF CURRENT SCREEN MASK
216
218
IS IT "ENTER" KEY ?
YES
NO
ENTER CHARACTER STRING FROM KEY INPUT BUFFER INTO CURRENTLY ACTIVATED FIELD/ PAGE OF SCREEN MASK AND ACTIVATE NEXT FIELD/PAGE TO THEREBY DEFINE RECEIVER ADDRESS INFORMATION OPTIONALLY INCLUDING PREFERRED SHIPPING MODE
220
222
IS IT "F10"-KEY ?
YES
NO
STORE RECEIVER INFORMATION FROM FIELDS OF SCREEN MASK IN INFORMATION BLOCK OF 1st STORAGE PORTION
224
226
IS IT "F5"-KEY ?
YES
NO
DISPLAY NEXT PAGE (S) OF RECEIVER INFORMATION MASK FOR ENTRY OF SUPPLEMENTARY INFORMATION
228

FIG. 4B

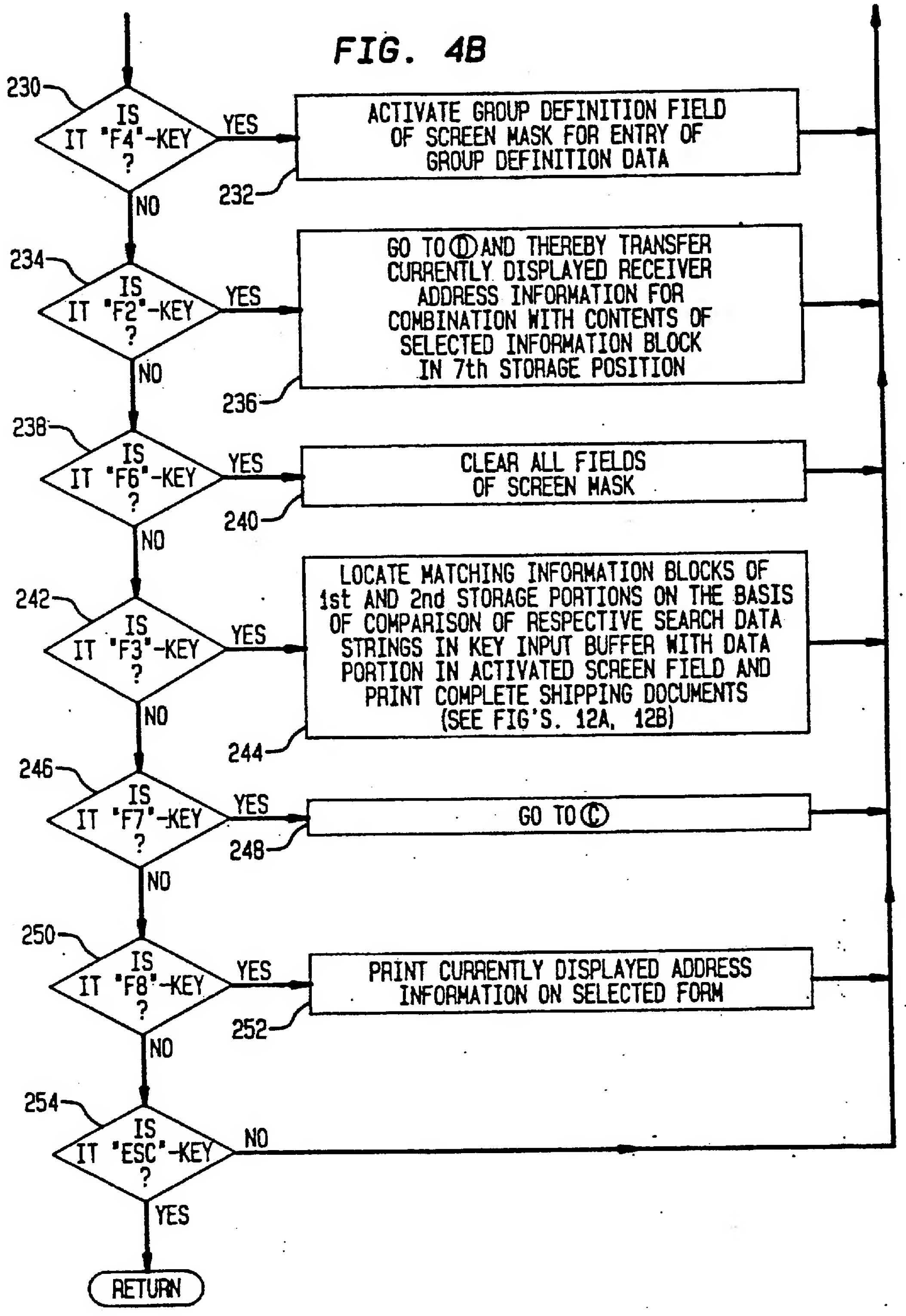
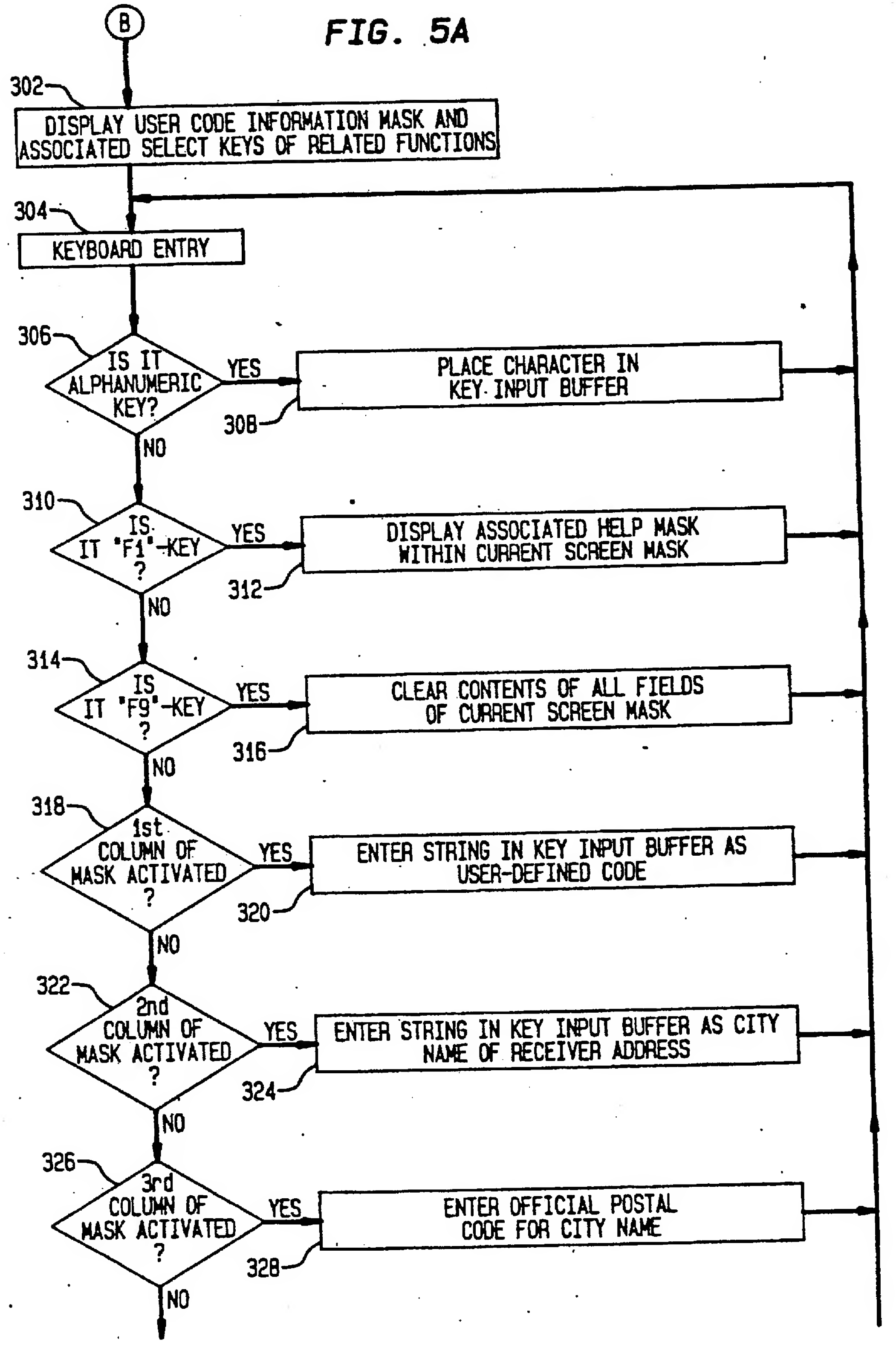

230
IS IT "F4"-KEY ?
YES
ACTIVATE GROUP DEFINITION FIELD OF SCREEN MASK FOR ENTRY OF GROUP DEFINITION DATA
232
NO
234
IS IT "F2"-KEY ?
YES
GO TO (D) AND THEREBY TRANSFER CURRENTLY DISPLAYED RECEIVER ADDRESS INFORMATION FOR COMBINATION WITH CONTENTS OF SELECTED INFORMATION BLOCK IN 7th STORAGE POSITION
236
NO
238
IS IT "F6"-KEY ?
YES
CLEAR ALL FIELDS OF SCREEN MASK
240
NO
242
IS IT "F3"-KEY ?
YES
LOCATE MATCHING INFORMATION BLOCKS OF 1st AND 2nd STORAGE PORTIONS ON THE BASIS OF COMPARISON OF RESPECTIVE SEARCH DATA STRINGS IN KEY INPUT BUFFER WITH DATA PORTION IN ACTIVATED SCREEN FIELD AND PRINT COMPLETE SHIPPING DOCUMENTS (SEE FIG'S. 12A, 12B)
244
NO
246
IS IT "F7"-KEY ?
YES
GO TO (C)
248
NO
250
IS IT "F8"-KEY ?
YES
PRINT CURRENTLY DISPLAYED ADDRESS INFORMATION ON SELECTED FORM
252
NO
254
IS IT "ESC"-KEY ?
NO
YES
RETURN

FIG. 5A


B
302
DISPLAY USER CODE INFORMATION MASK AND ASSOCIATED SELECT KEYS OF RELATED FUNCTIONS
304
KEYBOARD ENTRY
306
IS IT ALPHANUMERIC KEY?
YES
PLACE CHARACTER IN KEY INPUT BUFFER
308
NO
310
IS IT "F1"-KEY ?
YES
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
312
NO
314
IS IT "F9"-KEY ?
YES
CLEAR CONTENTS OF ALL FIELDS OF CURRENT SCREEN MASK
316
NO
318
1st COLUMN OF MASK ACTIVATED ?
YES
ENTER STRING IN KEY INPUT BUFFER AS USER-DEFINED CODE
320
NO
322
2nd COLUMN OF MASK ACTIVATED ?
YES
ENTER STRING IN KEY INPUT BUFFER AS CITY NAME OF RECEIVER ADDRESS
324
NO
326
3rd COLUMN OF MASK ACTIVATED ?
YES
ENTER OFFICIAL POSTAL CODE FOR CITY NAME
328
NO

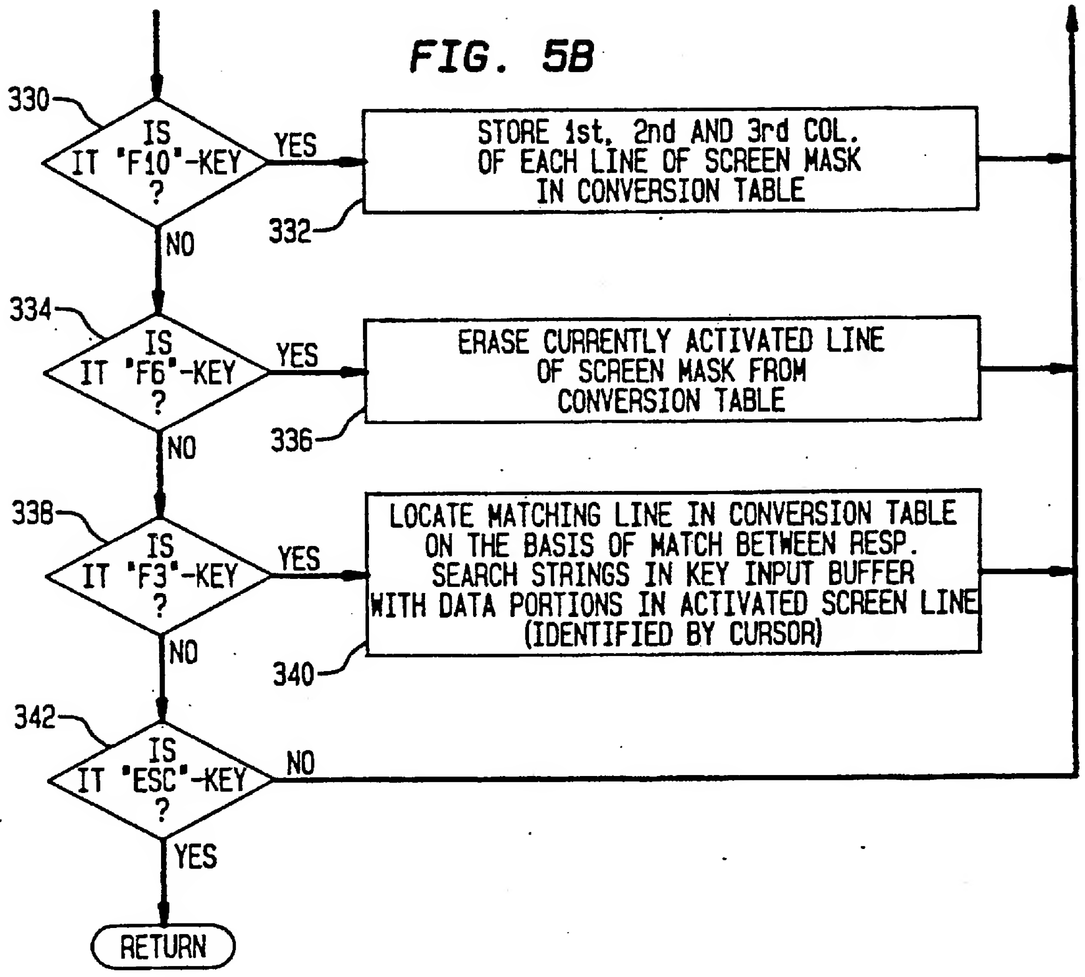

330
IS IT "F10"-KEY ?
YES
STORE 1st, 2nd AND 3rd COL. OF EACH LINE OF SCREEN MASK IN CONVERSION TABLE
332
NO
334
IS IT "F6"-KEY ?
YES
ERASE CURRENTLY ACTIVATED LINE OF SCREEN MASK FROM CONVERSION TABLE
336
NO
338
IS IT "F3"-KEY ?
YES
LOCATE MATCHING LINE IN CONVERSION TABLE ON THE BASIS OF MATCH BETWEEN RESP. SEARCH STRINGS IN KEY INPUT BUFFER WITH DATA PORTIONS IN ACTIVATED SCREEN LINE (IDENTIFIED BY CURSOR)
340
NO
342
IS IT "ESC"-KEY ?
NO
YES
RETURN

FIG. 5B

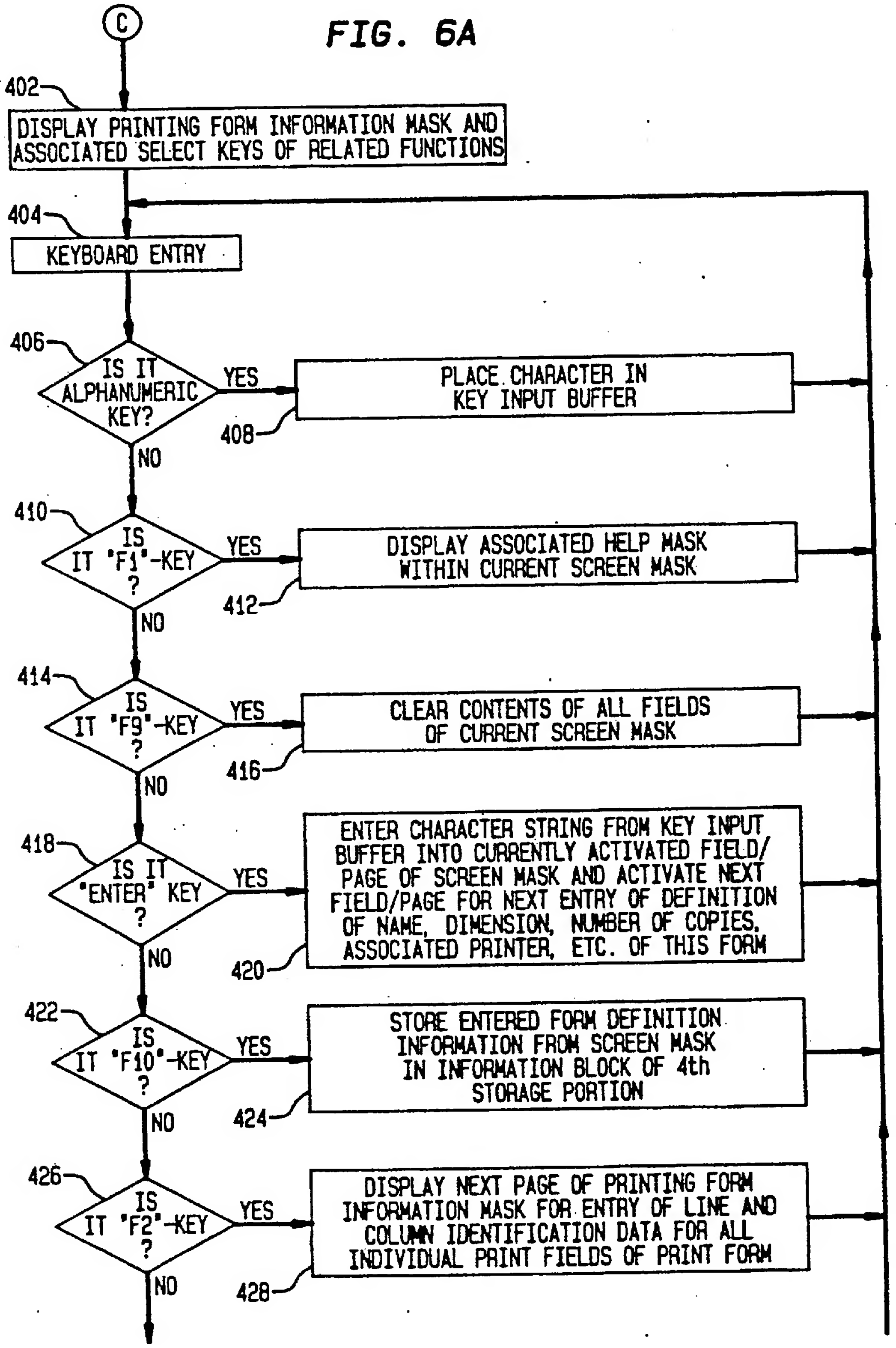

C
402
DISPLAY PRINTING FORM INFORMATION MASK AND ASSOCIATED SELECT KEYS OF RELATED FUNCTIONS
404
KEYBOARD ENTRY
406
IS IT ALPHANUMERIC KEY?
YES
NO
PLACE CHARACTER IN KEY INPUT BUFFER
408
410
IS IT "F1"-KEY ?
YES
NO
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
412
414
IS IT "F9"-KEY ?
YES
NO
CLEAR CONTENTS OF ALL FIELDS OF CURRENT SCREEN MASK
416
418
IS IT "ENTER" KEY ?
YES
NO
ENTER CHARACTER STRING FROM KEY INPUT BUFFER INTO CURRENTLY ACTIVATED FIELD/PAGE OF SCREEN MASK AND ACTIVATE NEXT FIELD/PAGE FOR NEXT ENTRY OF DEFINITION OF NAME, DIMENSION, NUMBER OF COPIES, ASSOCIATED PRINTER, ETC. OF THIS FORM
420
422
IS IT "F10"-KEY ?
YES
NO
STORE ENTERED FORM DEFINITION INFORMATION FROM SCREEN MASK IN INFORMATION BLOCK OF 4th STORAGE PORTION
424
426
IS IT "F2"-KEY ?
YES
NO
DISPLAY NEXT PAGE OF PRINTING FORM INFORMATION MASK FOR ENTRY OF LINE AND COLUMN IDENTIFICATION DATA FOR ALL INDIVIDUAL PRINT FIELDS OF PRINT FORM
428

FIG. 6A

FIG. 6B

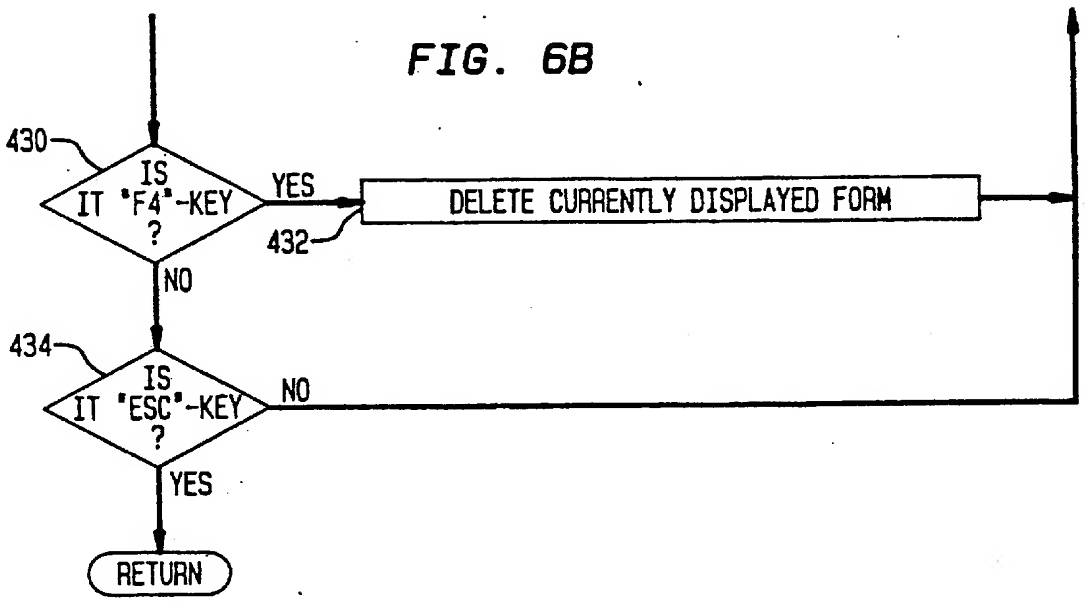

430
IS IT "F4"-KEY ?
YES
DELETE CURRENTLY DISPLAYED FORM
432
NO
434
IS IT "ESC"-KEY ?
NO
YES
RETURN

FIG. 7A

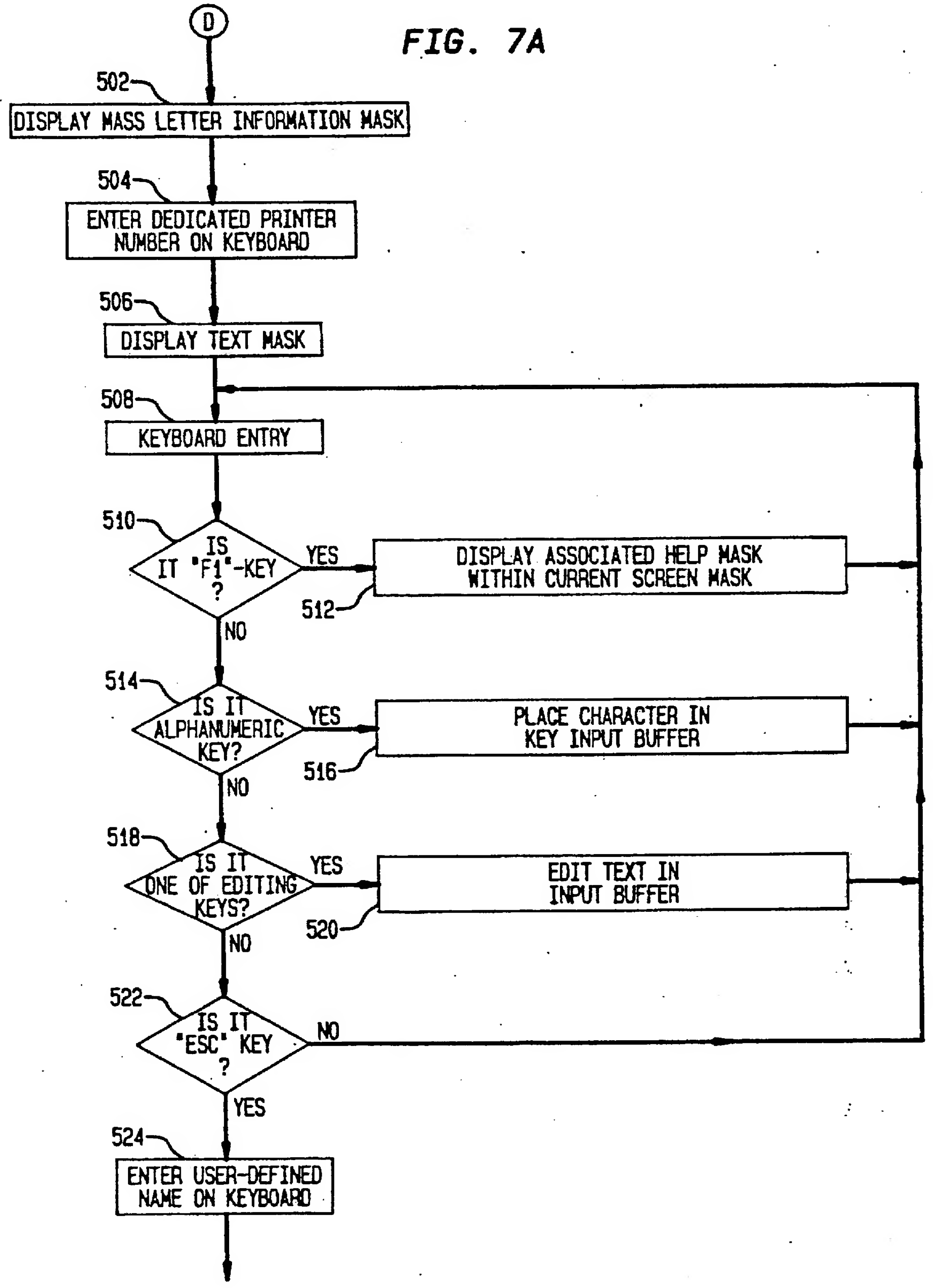

D
502
DISPLAY MASS LETTER INFORMATION MASK
504
ENTER DEDICATED PRINTER NUMBER ON KEYBOARD
506
DISPLAY TEXT MASK
508
KEYBOARD ENTRY
510
IS IT "F1"-KEY ?
YES
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
512
NO
514
IS IT ALPHANUMERIC KEY?
YES
PLACE CHARACTER IN KEY INPUT BUFFER
516
NO
518
IS IT ONE OF EDITING KEYS?
YES
EDIT TEXT IN INPUT BUFFER
520
NO
522
IS IT "ESC" KEY ?
NO
YES
524
ENTER USER-DEFINED NAME ON KEYBOARD

FIG. 7B

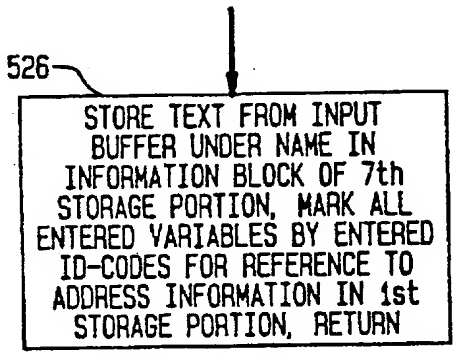

526
STORE TEXT FROM INPUT BUFFER UNDER NAME IN INFORMATION BLOCK OF 7th STORAGE PORTION, MARK ALL ENTERED VARIABLES BY ENTERED ID-CODES FOR REFERENCE TO ADDRESS INFORMATION IN 1st STORAGE PORTION, RETURN

FIG. 8A

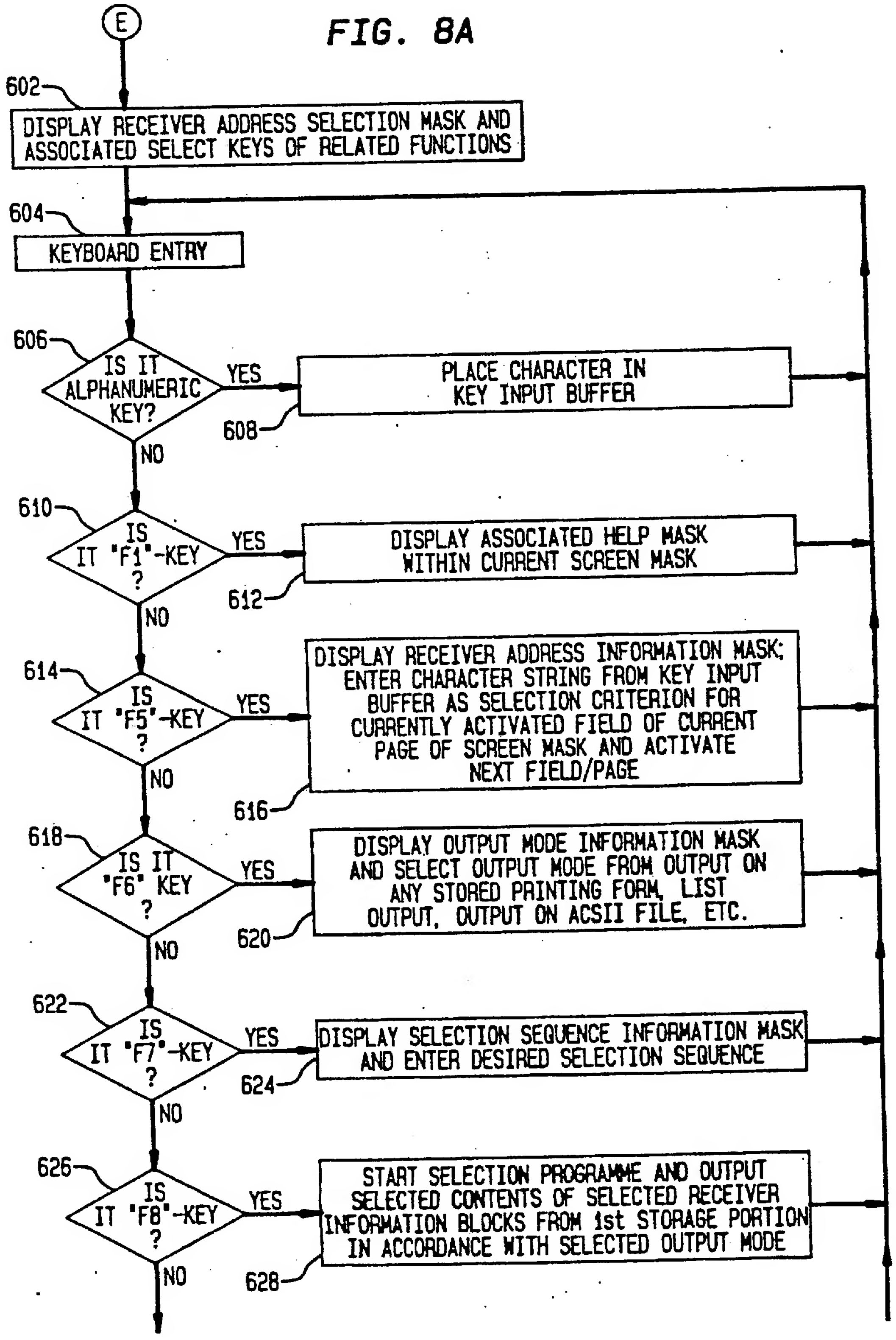

E
602
DISPLAY RECEIVER ADDRESS SELECTION MASK AND ASSOCIATED SELECT KEYS OF RELATED FUNCTIONS
604
KEYBOARD ENTRY
606
IS IT ALPHANUMERIC KEY?
YES
NO
PLACE CHARACTER IN KEY INPUT BUFFER
608
610
IS IT "F1"-KEY ?
YES
NO
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
612
614
IS IT "F5"-KEY ?
YES
NO
DISPLAY RECEIVER ADDRESS INFORMATION MASK; ENTER CHARACTER STRING FROM KEY INPUT BUFFER AS SELECTION CRITERION FOR CURRENTLY ACTIVATED FIELD OF CURRENT PAGE OF SCREEN MASK AND ACTIVATE NEXT FIELD/PAGE
616
618
IS IT "F6" KEY ?
YES
NO
DISPLAY OUTPUT MODE INFORMATION MASK AND SELECT OUTPUT MODE FROM OUTPUT ON ANY STORED PRINTING FORM, LIST OUTPUT, OUTPUT ON ACSII FILE, ETC.
620
622
IS IT "F7"-KEY ?
YES
NO
DISPLAY SELECTION SEQUENCE INFORMATION MASK AND ENTER DESIRED SELECTION SEQUENCE
624
626
IS IT "F8"-KEY ?
YES
NO
START SELECTION PROGRAMME AND OUTPUT SELECTED CONTENTS OF SELECTED RECEIVER INFORMATION BLOCKS FROM 1st STORAGE PORTION IN ACCORDANCE WITH SELECTED OUTPUT MODE
628

FIG. 8B

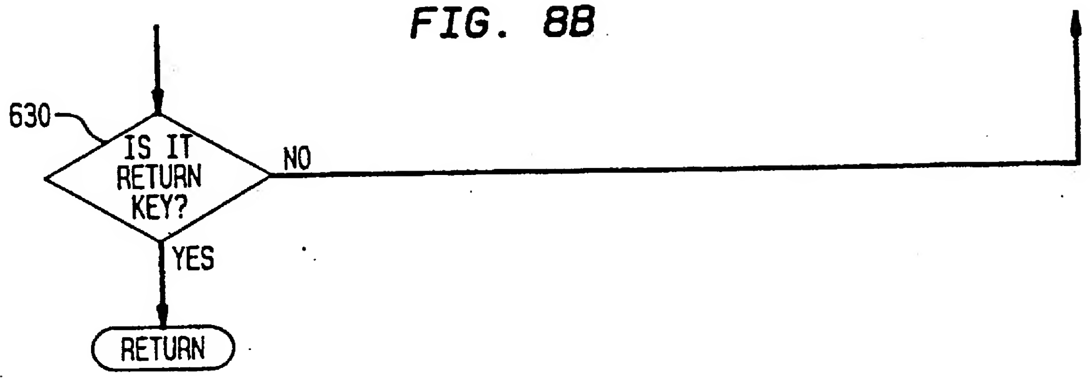


FIG. 9

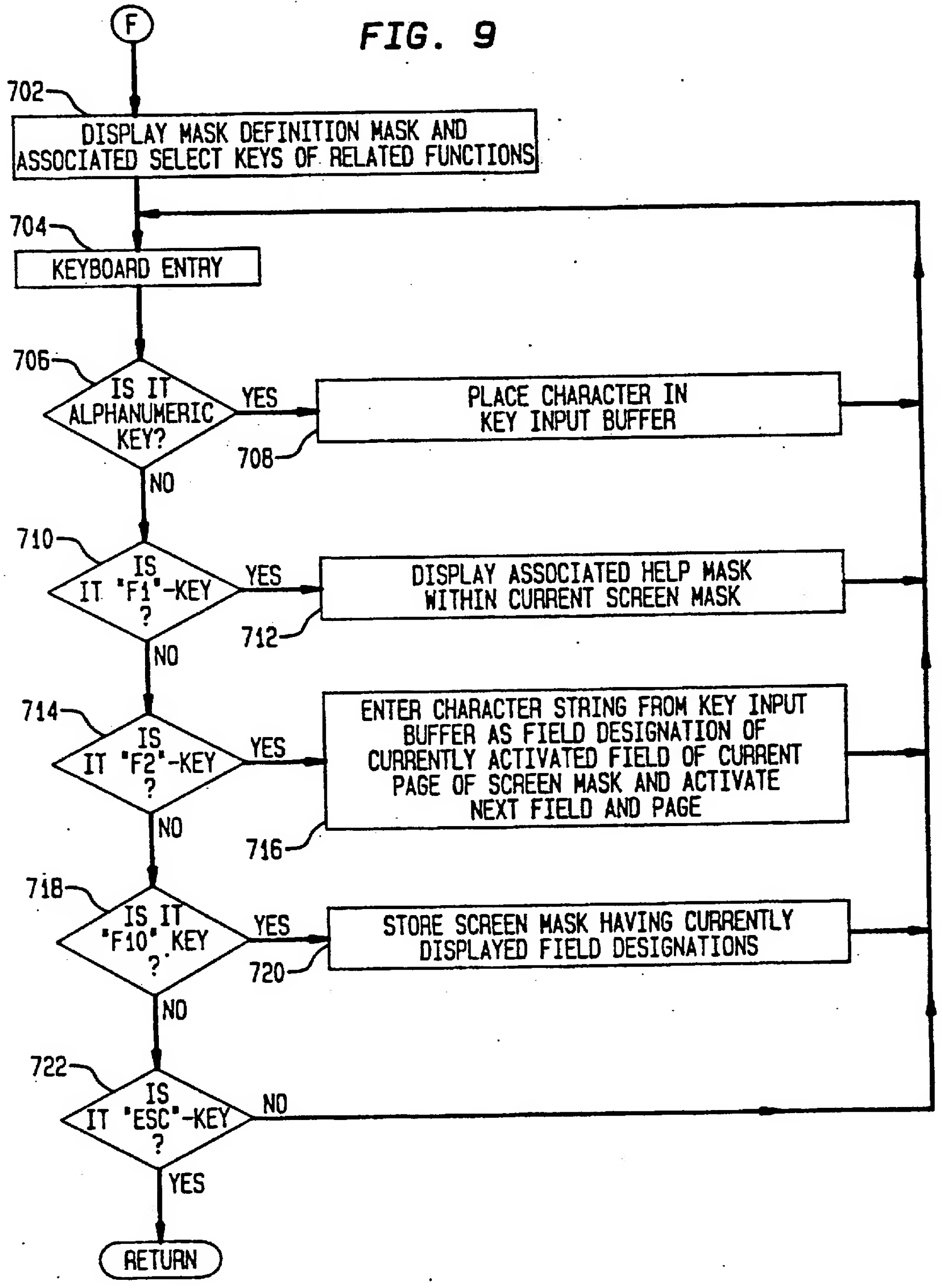
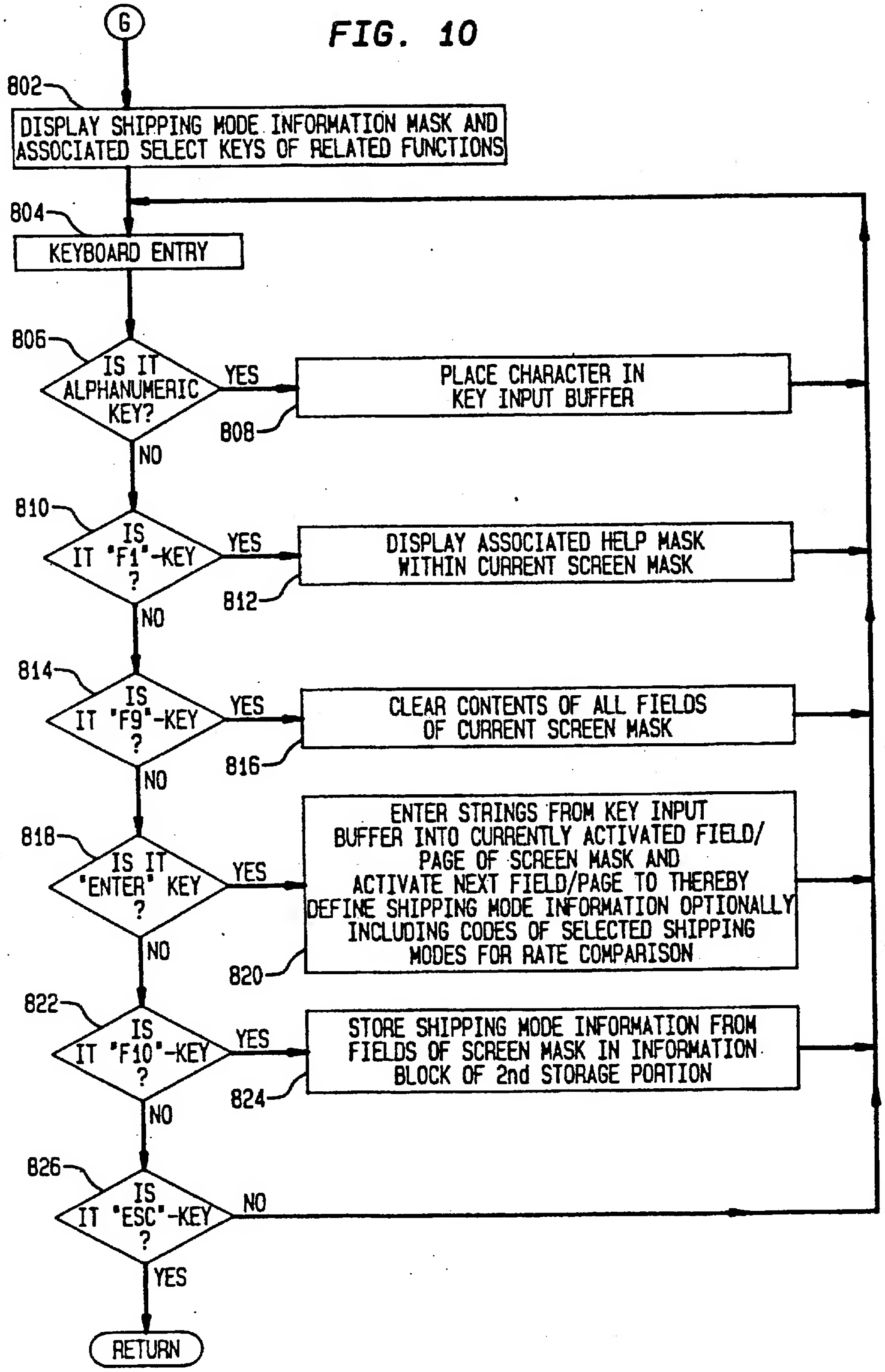

F
702
DISPLAY MASK DEFINITION MASK AND ASSOCIATED SELECT KEYS OF RELATED FUNCTIONS
704
KEYBOARD ENTRY
706
IS IT ALPHANUMERIC KEY?
YES
PLACE CHARACTER IN KEY INPUT BUFFER
708
NO
710
IS IT "F1"-KEY ?
YES
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
712
NO
714
IS IT "F2"-KEY ?
YES
ENTER CHARACTER STRING FROM KEY INPUT BUFFER AS FIELD DESIGNATION OF CURRENTLY ACTIVATED FIELD OF CURRENT PAGE OF SCREEN MASK AND ACTIVATE NEXT FIELD AND PAGE
716
NO
718
IS IT "F10" KEY ?
YES
STORE SCREEN MASK HAVING CURRENTLY DISPLAYED FIELD DESIGNATIONS
720
NO
722
IS IT "ESC"-KEY ?
NO
YES
RETURN

FIG. 10



G
802
DISPLAY SHIPPING MODE INFORMATION MASK AND ASSOCIATED SELECT KEYS OF RELATED FUNCTIONS
804
KEYBOARD ENTRY
806
IS IT ALPHANUMERIC KEY?
YES
PLACE CHARACTER IN KEY INPUT BUFFER
808
NO
810
IS IT "F1"-KEY ?
YES
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
812
NO
814
IS IT "F9"-KEY ?
YES
CLEAR CONTENTS OF ALL FIELDS OF CURRENT SCREEN MASK
816
NO
818
IS IT "ENTER" KEY ?
YES
ENTER STRINGS FROM KEY INPUT BUFFER INTO CURRENTLY ACTIVATED FIELD/PAGE OF SCREEN MASK AND ACTIVATE NEXT FIELD/PAGE TO THEREBY DEFINE SHIPPING MODE INFORMATION OPTIONALLY INCLUDING CODES OF SELECTED SHIPPING MODES FOR RATE COMPARISON
820
NO
822
IS IT "F10"-KEY ?
YES
STORE SHIPPING MODE INFORMATION FROM FIELDS OF SCREEN MASK IN INFORMATION BLOCK OF 2nd STORAGE PORTION
824
NO
826
IS IT "ESC"-KEY ?
NO
YES
RETURN

FIG. 11A

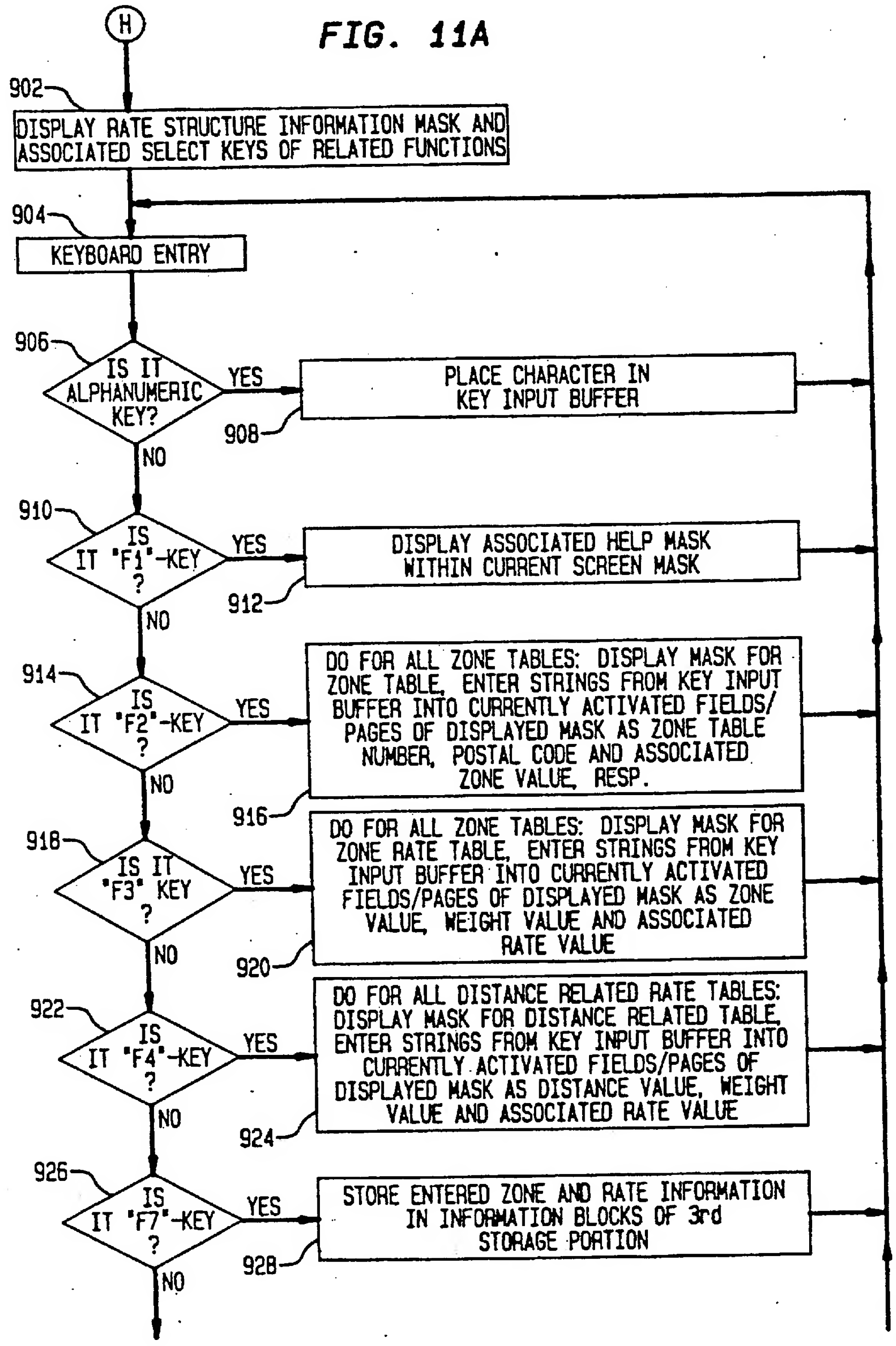

H
902
DISPLAY RATE STRUCTURE INFORMATION MASK AND ASSOCIATED SELECT KEYS OF RELATED FUNCTIONS
904
KEYBOARD ENTRY
906
IS IT ALPHANUMERIC KEY?
YES
NO
PLACE CHARACTER IN KEY INPUT BUFFER
908
910
IS IT "F1"-KEY ?
YES
NO
DISPLAY ASSOCIATED HELP MASK WITHIN CURRENT SCREEN MASK
912
914
IS IT "F2"-KEY ?
YES
NO
DO FOR ALL ZONE TABLES: DISPLAY MASK FOR ZONE TABLE, ENTER STRINGS FROM KEY INPUT BUFFER INTO CURRENTLY ACTIVATED FIELDS/PAGES OF DISPLAYED MASK AS ZONE TABLE NUMBER, POSTAL CODE AND ASSOCIATED ZONE VALUE, RESP.
916
918
IS IT "F3" KEY ?
YES
NO
DO FOR ALL ZONE TABLES: DISPLAY MASK FOR ZONE RATE TABLE, ENTER STRINGS FROM KEY INPUT BUFFER INTO CURRENTLY ACTIVATED FIELDS/PAGES OF DISPLAYED MASK AS ZONE VALUE, WEIGHT VALUE AND ASSOCIATED RATE VALUE
920
922
IS IT "F4"-KEY ?
YES
NO
DO FOR ALL DISTANCE RELATED RATE TABLES: DISPLAY MASK FOR DISTANCE RELATED TABLE, ENTER STRINGS FROM KEY INPUT BUFFER INTO CURRENTLY ACTIVATED FIELDS/PAGES OF DISPLAYED MASK AS DISTANCE VALUE, WEIGHT VALUE AND ASSOCIATED RATE VALUE
924
926
IS IT "F7"-KEY ?
YES
NO
STORE ENTERED ZONE AND RATE INFORMATION IN INFORMATION BLOCKS OF 3rd STORAGE PORTION
928

FIG. 11B

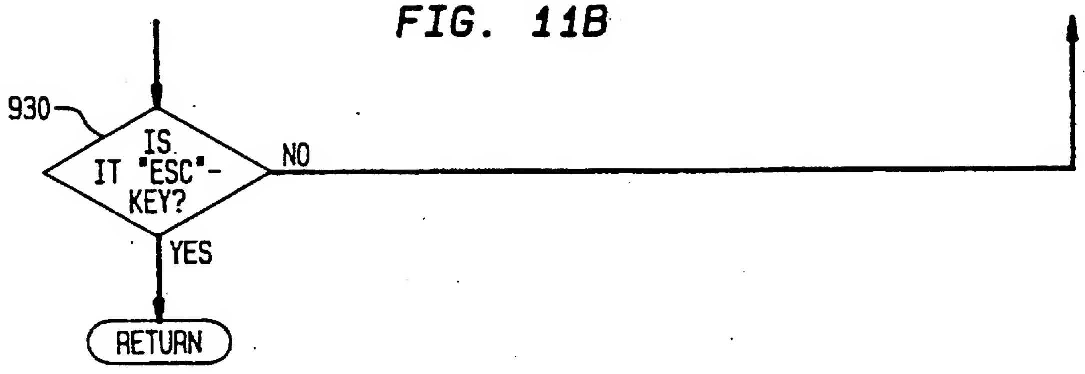


FIG. 12A

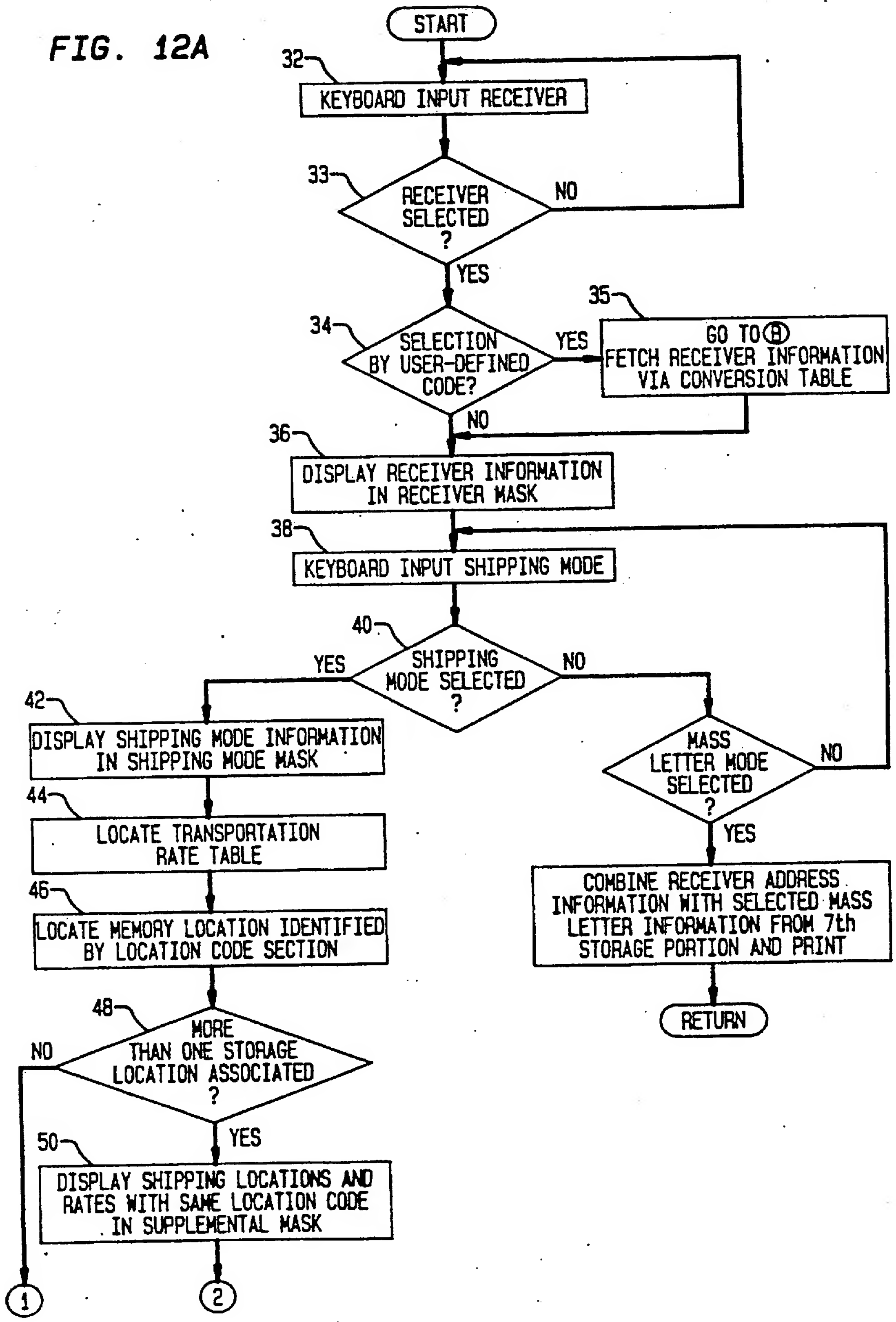
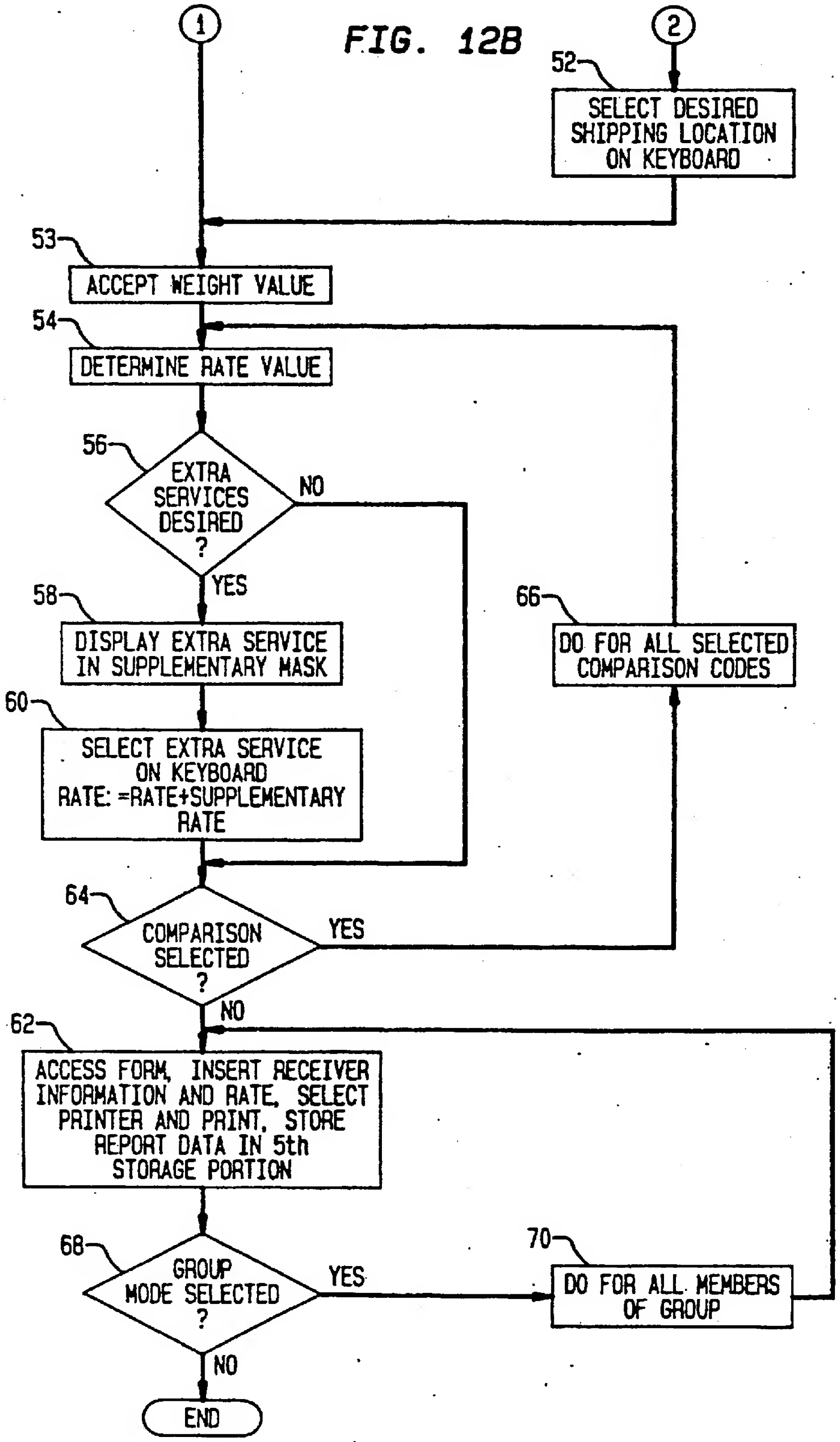

START
32 KEYBOARD INPUT RECEIVER
33 RECEIVER SELECTED ?
NO
YES
34 SELECTION BY USER-DEFINED CODE?
YES
35 GO TO B FETCH RECEIVER INFORMATION VIA CONVERSION TABLE
NO
36 DISPLAY RECEIVER INFORMATION IN RECEIVER MASK
38 KEYBOARD INPUT SHIPPING MODE
40 SHIPPING MODE SELECTED ?
YES
NO
42 DISPLAY SHIPPING MODE INFORMATION IN SHIPPING MODE MASK
MASS LETTER MODE SELECTED ?
NO
YES
44 LOCATE TRANSPORTATION RATE TABLE
COMBINE RECEIVER ADDRESS INFORMATION WITH SELECTED MASS LETTER INFORMATION FROM 7th STORAGE PORTION AND PRINT
46 LOCATE MEMORY LOCATION IDENTIFIED BY LOCATION CODE SECTION
RETURN
48 MORE THAN ONE STORAGE LOCATION ASSOCIATED ?
NO
YES
50 DISPLAY SHIPPING LOCATIONS AND RATES WITH SAME LOCATION CODE IN SUPPLEMENTAL MASK
1
2

FIG. 12B


1
2
52
SELECT DESIRED SHIPPING LOCATION ON KEYBOARD
53
ACCEPT WEIGHT VALUE
54
DETERMINE RATE VALUE
56
EXTRA SERVICES DESIRED ?
NO
YES
58
DISPLAY EXTRA SERVICE IN SUPPLEMENTARY MASK
60
SELECT EXTRA SERVICE ON KEYBOARD RATE: =RATE+SUPPLEMENTARY RATE
66
DO FOR ALL SELECTED COMPARISON CODES
64
COMPARISON SELECTED ?
YES
NO
62
ACCESS FORM, INSERT RECEIVER INFORMATION AND RATE, SELECT PRINTER AND PRINT, STORE REPORT DATA IN 5th STORAGE PORTION
68
GROUP MODE SELECTED ?
YES
70
DO FOR ALL MEMBERS OF GROUP
NO
END

SYSTEM FOR PREPARING SHIPPING DOCUMENTS

This application is a continuation of application Ser. No. 323,760, filed Mar. 15, 1989, now abandoned.

BACKGROUND OF THE INVENTION

Reference is made to pending patent application number P 38 08 616.6, filed in West Germany on Mar. 15, 1988 by Pitney Bowes Deutschland, GMBH, the employer of the applicants, which application realtes to the same invention as this application and under which priority is claimed pursuant to 35 U.S.C., section 119. The invention relates to a system for processing articles for shipment, including a scale, a printer, a memory and a keyboard, and a processor coupled to these components.

Previously known systems have been used to prepare shipping documents, such as parcel labels, waybills, manifests, and the like. Although these systems have simplified to some extent the steps involved in preparing such documents, these systems were not capable of preparing the documents required for various shipping modes, such as mail, train, air freight or private carriers without overnight express, parcel post, air or surface freight, or private or public carriers, without substantial and time-consuming efforts by an operator. In addition, different rate tables, calculating modes and form requirements have to be observed. It should be noted that the various carriers each require documents that differ from one another in format and information required.

SUMMARY OF THE INVENTION

The invention relates to a system for automatically preparing documents for shipping articles to any desired number of different receivers by any selected shipping mode.

The system includes weight data entry means, a printer, memory, a keyboard and a processor. The weight data entry means, which is capable of causing the printer to print more than one type of waybill. The waybill as printed reflects the data received by the processor from the various components. Typically the system operates in the following manner:

The user inputs the weight of the article to be shipped, perhaps by placing the article on a scale, which is in communication with the processor. Through the keyboard, the user can select the recipient of the article from among those names and addresses that are stored in memory and retreivable through the keyboard. Next, again using the keyboard, the user selects the desired carrier and mode of shipment. The system then prints a waybill and, if necessary, other documents that the carrier requires to be submitted with the article to be shipped. The system according to this invention is more versatile than prior systems in that it is capable of printing a wide variety of documents required by carriers. Also, in its various embodiments, the system provides convenient features for the user that were not previously available.

DESCRIPTION OF THE DRAWING

FIG. 1 shows a schematic diagram of a system for preparing shipping documents.

FIG. 2 (*a*) to (*g*) shows the structure of information blocks provided in storage portions of storage means in an embodiment of said system.

FIG. 3 (*a*) and (*b*) a schematic flow chart of a main routine for branching into various operational routines as provided for in an embodiment of the system.

FIG. 4 (*a*) and (*b*) a schematic flow chart of a routine for storing recipient information.

FIG. 5 (*a*) and (*b*) a schematic flow chart of a routine for storing user-defined location codes.

FIG. 6 (*a*) and (*b*) a schematic flow chart of a routine for storing printing form definition information.

FIG. 7 (*a*) and (*b*) a schematic flow chart of a routine for mass letter processing.

FIG. 8 (*a*) and (*b*) a schematic flow chart of a routine for selecting and processing a group of receiver addresses selected from the stored receiver addresses.

FIG. 9 a schematic flow chart of a routine for user definition of mask format.

FIG. 10 a schematic flow chart of a routine for storing shipping mode information.

FIG. 11 (*a*) and (*b*) a schematic routine for storing of transportation rate information, and

FIG. 12 (*a*) and (*b*) a schematic flow chart of the control flow of the system in an automatic shipping operation with automatic access to a transportation rate table.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

A system for preparing shipping documents is schematically represented in FIG. 1 and has a processor 1 connected to a keyboard 2, a scale 3 for weighing the articles to be shipped, and a printer 4. The processor is also connected to a visual display 5 and a memory 6, which is addressed by the processor 1 for random access in read and/or write operations. The processor 1 further includes an input/output interface 7 for connection with an external data source such as a host computer. The processor 1, operating under the control of a program, performs a number of functions as described below.

The memory 6 includes first, second, third, fourth, fifth, sixth and seventh storage portions (8, 9, 10, 11, 12, 19 and 21 respectively), which are accessed for various purposes by the processor while it performs its functions.

The first storage portion 8 of the memory 6 contains a desired first number of information blocks 13. The structure of the information blocks 13 are symbolically illustrated in FIG. 2 (*a*). As may be seen therefrom, the information block 13 has two associated blocks of storage locations, one of the blocks representing a recipient address text section for storing the recipient's address and the other block representing a location code section for storing a code that identifies the geographic location of the recipient.

The recipient address section includes other information such as a salutation, the name of the recipient and the address. The same section also includes a storage portion for storing additional alphanumeric information regarding the recipient, a section for storing a code representative of the recipient's preferred shipping mode and a section for storing group definition data that associates the recipient with a selected one of several groups. The recipient information which is stored in the information blocks of the first storage portion need not be restricted to mere addressing information like name, street, city and country or the like, but could include other information concerning customers. For this purpose, each of the information blocks in said first storage portion has a supplementary information section adapted for storing additional information other than recipient address information. Such additional information can include the products customarily bought by this recipient, prices, and sales history, such as a day-by-day history for current year and total for previous year.

The zip code corresponding to the receiver location can be contained in the location code section. Alternatively, the location code section can contain a code defined by the user that specifies a single recipient location. As will be discussed below, such a code enables the system to ascertain the appropriate rate for a shipment to that location even when the zip code has more than one rate zone.

A desired number of information blocks **14** is contained in the second storage portion **9** of the memory **6** as shown in FIG. 2 (*b*). The user may define an information block **14** for each shipping mode of each carrier the user desires to engage from time to time. The desired shipping modes can include overnight air express, parcel post, and air and surface freight. Each information block **14** contains three associated blocks of storage locations, one of these blocks representing a shipping mode text section for storing text data to identify a predetermined shipping mode, another block representing a rate code section for storing pointer information to a rate table, and the third block representing a form code section for storing pointer information to one or several predetermined forms.

The shipping mode text section contains all data necessary for identifying a predetermined shipping mode of a predetermined carrier.

In one embodiment of the invention, the user can define code values identifying each one of the available shipping modes. Each code value is stored in a shipping mode text section together with the code values of such other shipping modes as the user may select. As will be discussed below, this makes it possible for the user to do "rate shopping", i.e., to compare the amounts that would be changed by a number of carriers for transporting a given parcel to a given location. For each shipping mode there is a predetermined transportation rate charge table from which the charge is determined for shipping an article of a given weight a certain distance or to a certain zone. In the rate code section, pointer information is stored which identifies the applicable transportation rate table for the shipping mode identified by the information block **14**.

For each shipping mode, the carrier requires that certain forms be submitted with the article to be shipped. These forms may include a prescribed way bill, a manifest or an address label. In the form code section pointer information is stored for identifying which and how many forms are required.

The transportation rate tables identified by the rate code sections of the information blocks **14** of the second storage portion **9** are each stored in information blocks **15** of the third storage portion **10**.

According to FIG. 2 (*c*) these information blocks **15** contain, in a table section, storage locations which are addressable by (*a*) the contents of the location code section of the information block **13** (representing the zone of, or the distance to, the destination), and (*b*) the weight value of the article to be shipped, in order to enable read-out of the correct transportation rate value for every transported weight and every transportation distance or zone.

For example, in some countries public carriers like postal services or train services use zone-structured rate tables. For every sender location a zone conversion table is available from the carrier. This zone conversion table associates a unique distance zone number, e.g. numbers 1, 2, 3 . . . up to some finite number n, to any receiver location as a function of its postal code. The processor then has to select this zone conversion table upon entry of the desired carrier and desired shipping mode and to determine the appropriate zone number based upon the receiver location postal code. With this zone number, the processor may then determine the transportation rate value as a function of the entered weight by reference to the applicable transportation rate table.

Some carriers make use of distance-related rate tables. For every sender location a table is available from which the shipping distance between the sender location and any receiver location may be read out, while the receiver location is defined in terms of its postal code and optionally an extension code discriminating between certain subareas within the area defined by the postal code. In the latter case manual or automatic conversion, as described in more detail below, has to be performed in order to determine the distance value on the basis of postal code. The distance value determined in this way defines the address for the table containing the applicable transportation rate values as a function of the weight of the parcel.

One or more of the information blocks **15** may include, according to FIG. 2 (*c*), an associated supplementary information section in which data corresponding to optional extra services and supplementary rates are stored. For instance the extra service "express transport" may be provided for a certain shipping mode and the normal transportation rate value has to be augmented by a fixed supplementary rate value.

The printing formats identified by the form code section of the information blocks **14** of the second storage portion **9**, are according to FIG. 2 (*d*), in a desired fourth number of information blocks **16** stored in the fourth storage portion **11**. Each information block **16** contains the control commands and text information data required for causing the printer to print out the form or forms required for a certain shipping mode. In the data stored in the information blocks **16** there are empty locations at the appropriate positions in which there is inserted prior to print out information necessary for completion of the form. That information is obtained from the information blocks **13**, **15**, and, if applicable **14**.

FIGS. 12 (*a*) and (*b*) illustrate the operation of the system when it is issued to prepare shipping documents. The process begins with selection of the desired recipient. This can be accomplished by entry via the keyboard of the recipient's name (steps **34** and **35**) or address (steps **32** and **33**) or an identifying code (steps **34** and **35**) or entry of only a portion of one of those items.

Alternatively, the user may through the keyboard commence search operation, under control of a certain key of the keyboard **2**, whereby the individual information blocks **13** are searched through one after the other to locate the information block **13** containing the data for the desired recipient.

In either case, once a valid recipient information block **13** has been selected the information contained in the information block **13** is displayed on the display embedded in an appropriate mask (step **36**), as further discussed below. Then the keyboard 2 is enabled to select the desired shipping mode (step 38). This, can be accompanied by entry of an appropriate portion of the carrier name, or of a code therefore. Retrieval of the desired information block 14 may also be provided by successive search through all information blocks 14 provided in the second storage portion 9.

In response to the selection of the shipping mode, (step 40), the processor operates the display 5 so that it displays the shipping mode contained in the information block 14 (step 42). The processor simultaneously caused the display to show an appropriately designed mask on the screen which may for instance be called by access to a further storage portion 17 of the memory 6. In this way the user gets a clearly arranged and easily understandable visual representation of the information contents selected by him via the keyboard 2, since the masks include a format and supplementary text information that explains the selected information. The same use of masks occurs in steps 32–36, referred to above.

Moreover, in one embodiment of the invention the processor may be programmed to allow the user to alter the mask formats via the keyboard. This embodiment allows the user to customize the masks for the user's specific applications.

The system further incorporates help functions pertaining to the displayed screen masks. When the user activates a help key on said keyboard the help function is called and within a portion of the displayed mask instructional information on the further steps to be performed and the types of data that may be entered. The currently activated information line or block of the currently displayed screen mask is sensed by the processor, which then causes the display to show instructional information related to that currently activated information line or block.

On the basis of the rate code section read out from the information block 14 of the second storage portion 9 the processor 1 accesses from the third storage portion 10 the specific information block 15 identified by that rate code section (step 44). Then the process retrieves from that information block 15 the memory location therein identified by the contents of the location code section (step 46) then checks whether the contents of the read-out location code section is associated with one or with more than one memory locations of the transportation rate table (step 48). For example, this may be accomplished by storing those memory locations of the transportation rate table which are associated to one and the same location code in sequence one after the other. This allows the processor, after retreival of the first memory location corresponding to the location code section, to check the immediately following memory location in order to determine whether there are further memory locations for the same location code. If the location code associated with this memory location is different, then only one memory location in the rate table is associated with that location code. Otherwise, the processor continues to the next memory location, and if necessary the next, and so forth, until a different location code is found.

Then the processor inserts a supplementary mask into the currently displayed screen mask. The supplementary mask contains text to inform the user that more than one recipient location is identified by the location code. Also, there is displayed within that supplementary mask, the various recipient locations and associated transportation rate values corresponding to the same

location code (e.g. to the same zip code) (step 50). Simultaneously the keyboard 2 is enabled to select the desired location (step 52).

The before-described manual selection between two or more memory locations storing transportation rate tables may occur when the postal code is used as location code while the selected shipping mode has a transportation rate structure with transportation rate values for different subareas of a single postal code area. Manual discrimination between the various subareas may, however, be avoided if the user predefines appropriate location codes as described hereinbefore. Such a user-defined code may, for instance, include the postal code supplemented by an extension for distinguishing the subareas. Having thus created a code which implies a one-to-one correspondence between the recipient addresses as identified by recipient location and subarea and the corresponding rate table entries, no manual discrimination is necessary and the system automatically accesses the applicable transportation rate table. The branch of control flow as represented by step 48 in FIG. 3 is then no longer required.

After the memory location corresponding to the desired destination location including the subarea identification, if any, has been determined, the processor 1 accepts the weight value of the piece to be dispatched from the scale 3 (step 53) and makes access to the cell within the corresponding memory location which is identified by the weight value in order to thereby read out the applicable transportation rate value (step 54).

In the embodiment illustrated in FIG. 1 the weight data entry means is scale 3 which automatically transmits the weight value to the processor 1. In other embodiments the weight data entry means may be an optical scanner to read weight information in the form of a bar code applied to the piece to be dispatched. It is also possible to program the processor 1 so that a numerical key section of the keyboard means 2 is enabled to input the weight.

As may be further seen from FIG. 12(*b*), the processor 1 now determines whether the information block 15 of the third storage portion 10 as accessed by the hitherto performed operating sequence comprises a supplementary information section. These data relating to optional extra services like express delivery, night delivery or the like and to the supplementary rate values applicable therefore are displayed by the processor 1 in a supplementary mask (step 58) which is inserted into the currently displayed screen mask and provides an explanation if the user indicates via keyboard 2 that an extra service is desired (step 56). Then the processor 1 enables the keyboard 2 to permit the user to select the desired extra service (step 60). After the user has selected the desired extra service in this way the corresponding supplementary rate value is added to the ordinary transportation rate value as determined according to the operating sequence described above.

As shown in step 62, on the basis of the memory signal read out from the form code section of the information block 14 the processor 1 means now selects the one or more information blocks 16 of the fourth storage portion 11 identified thereby and inserts the information concerning the recipient and the transportation rate value and optionally concerning the selected shipping mode into empty locations provided in the forms defined by the information blocks 16. Because of the association between shipping mode and form effected by the form code section it is assured that the correct form or forms for the desired shipping mode are always selected. The information which is thereby arranged in the applicable form format is then output by the processor 1 to the printer means 4 for print-out.

Also to be noted are steps 64 and 66, which respectively represent selection and execution of the rate shopping function described below. In addition, steps 68 and 70 respectively represent selection and execution of the group mode defined below.

By way of example, this invention may be embodied in a personal computer such as the NCR Model PC 810.

The printer 4 used in the illustrated embodiment preferably is capable of printing all the forms discussed in this specification and of operating in conjunction with a feeder for endless forms, a batch feeder and a feeder for single forms. A printer capable of performing in this manner, when controlled by the program that is part of this invention, is a Printer Model #7200, available from Juki Corporation, Chofu-shi, Tokyo, Japan. The Model 7200 printer includes a feeder for endless forms and a feeder for single forms. A batch feeder that operates with the Model 7200 is Model 7201 AMF, also available from Juki Corporation.

Depending on the memory contents of the form code section read out in accordance with the processing steps explained above, the processor 1 automatically selects the one or more feeders corresponding to the one or more required forms in sequence so that the forms may automatically be produced even though different feeders are required in view of the different physical properties of the forms.

Under the operating sequence described above the steps required for preparing shipping documents to any desired number of recipients in any desired shipping modes are minimized and simplified. Additionally the operating sequence comprises further functions in addition or ancillary to those represented in FIGS. 12 (*a*) and (*b*).

The report information is stored in said fifth storage portion 12 in each of information blocks 18 as symbolically represented in FIG. 2(*e*). Each information block 18 comprises a set of data relating to one individual executed shipping operation. This set of data comprises at least selected items from the memory contents of the recipient address text sections and the shipping mode text sections together with the related transportation rate value. The set of data also includes a parcel number which is automatically generated by the system for any executed shipping operation. Moreover, the calendar date and optionally day-time of the shipping operation is appended to this data set. For this purpose the system comprises real time generator means in order to automatically supply the calendar date and/or day-time for use with said protocol data. The set of data may also if the user desires, include additional information that the user has caused to be stored with the recipient's address.

The report information may be accessed via the keyboard 2 by the calendar date or by an interval of calendar dates and/or the recipient address and/or the shipping mode. Thus activity reports for all or selected ones from the group of all receivers or from all permissible shipping modes may be obtained for a desired interval, e.g. on a month-by-month, week-by-week or day-by-day basis or for whatever period the user desires. These reports may be displayed on the display 5 or may be output for printing or further processing. The processor is programmed to allow the user to define via the keyboard 2 both content and format of the desired reports.

A further function of the processor 1 which may be called by actuation of the keyboard 2 allows the user to obtain, for the purpose of comparison, the rates applicable for a variety of shipping modes, given a parcel's weight and its destination. This function is commonly known as "rate shopping". In this function, the processor determines the transportation rate value for the same recipient's address for alternative shipping modes and then displays the alternative transportation rate values in a corresponding screen mask. By actuation of the keyboard 2 the user may then select a preferred shipping mode from among the displayed alternatives, causing the printing of the documents required for that preferred shipping mode.

For the purposes of the rate shopping, a unique code value is assigned to each shipping mode as defined in the information blocks 14 and a set of code values is stored by the user in each information block 14, said set of code values determining for each shipping mode, those other shipping modes which are to be displayed for the purpose of comparison.

Thus the user is free to define for each shipping mode the other modes he wishes to see for rate shopping purposes. This is a more flexible rate shopping function than has been found in prior systems. In prior systems, shipping modes were grouped for rate shopping purposes, so that the same group of modes was displayed for comparison with each of the modes in that group. In the system of this invention, the user is free to define a unique comparison group for each mode. To illustrate this difference, in the present invention, the user may arrange, for instance that modes B, C and D be compared to mode A; modes C, D and E be compared to mode B, and modes D, E and F be compared to mode C. By contrast, in prior systems, the user was required to define groups, so that, assuming a defined group was A, B, C and D;

B, C and D would be compared to A
A, C and D would be compared to B
A, B and D would be compared to C; and
A, B and C would be compared to D

without any flexibility. Further, if the user desired also to compare mode E to mode D, he was required to add E to the group, which meant that E would also be compared to modes A, B and C, even though that was not desired.

The system is capable of performing various calculations including a calculation of total amounts of transportation fees for all or selected groups of recipients or for selected ones out of the available shipping modes, i.e. available carriers. The system is also able to calculate the cubic volume of the pieces to be dispatched from the dimensions thereof.

There exist shipping modes in which the pieces to be dispatched utilize reusable packing material, e.g. reusable containers or pallets. In order to account for the issuing and return of these reusable packing materials a sixth storage portion 19 is provided in the memory means 6. A desired sixth number of information blocks 20 in said sixth storage portion 19 contains data for keeping track of each item of reusable packing material used in each shipping operation. Each set of such data includes at least a serial number for the packing material to be entered via the keyboard means 2 together with recipient address information that is automatically transferred from the corresponding information block 13 in the first storage portion 8 when the system prepares the shipping documents. Other information such as the type of packing material and the calendar date of issue and return may be included in such data.

Thus the status of reusable packing material for each individual shipping operation is stored in one information block 20. To access the information the system retrieves an individual information block 20 in response to entry of identification data via the keyboard 2. This identification data may, for instance, be the serial number or the recipient's name. Optionally retrieval may be effected for a certain issue date or interval of issue dates or the like. The data sets may be accessed individually or in groups and may be displayed on the display 5 or printed by the printer 4.

In the typical operation of the system as described above the user enters data to identify an individual recipient and a desired shipping mode. However, in certain applications it is desirable to send identical items to a predetermined group of recipients. For example, in operations such as book clubs, defined groups of subscribers are to receive identical parcels. The system therefore allows for storing of group definition data in each of the information blocks 13 containing the recipient addresses. The user is then only required to enter via the keyboard the corresponding group definition data together with the desired shipping mode and the weight value of the parcel 2 and the system then automatically prepares the required shipping documents for all recipients in the seclected group. Alternatively, the weight value may be entered by placing the typical parcel on the scale.

In the operating modes as described above it is assumed that all information with respect to recipient addresses, shipping modes, transportation rate structures and forms has previously been stored in the memory 6 and thus is accessible in the processor's first operating mode. The process may, however, by means of an appropriate actuation of the keyboard 2 be put into a second operating mode in which data entered by the keyboard means 2 may selectively be stored into the information blocks 13 to 16 of the storage portions 8 to 11. Thereby it is possible at any time to change, supplement or update such information via the keyboard 2. This process is most useful for changes and/or updates of a moderate extent. When substantial amounts of new or additional information must be input an advantageous feature of the invention is the input/output interface 7 connected to the processor 1. By means of the interface any desired storage portion may be programmed with data from an external data source. The external data source may for instance be a central computer. Use of such an external data source allows a large amount of information to be entered within a short time. This is especially advantageous in applications with a large number of recipient addresses.

When programming the first storage portion 8 with recipient address information from an external data source like a central computer it is especially advantageous simultaneously to include with recipient's address, information indicative of the recipient's preferred mode of shipment and any preferred special services. To enable the information blocks 13 of the first storage portion 8 to accommodate such information an extra code section is adapted for storing a code representing the preferred shipping mode and optionally a further code representing a preferred special service such as express shipping. With the address information supplemented by the coded information of customer's preferred shipping mode and/or preferred special service, the system may then automatically access the preferred shipping mode and/or preferred special service when access is made to the receiver address by the keyboard means 2. The transportation rate value is then calculated and all necessary shipping documents are prepared by the system on the basis of the preferred shipping mode and preferred special service without any need for the user to manually enter a shipping mode and/or special service.

The system also includes output interface means for communicating the information contents of at least the information blocks 13 in said first storage portion 8 to an external computer. For example, sales information contained in the information blocks 13, including recipient address, and even goods purchased and the prices thereof, as well as transportation changes may be transferred to another computer for automatic creation of invoices.

A seventh storage portion 21 optionally provided in the memory means 6 may include a seventh number of information blocks 22 for storing mass letter information. The system allows for selection of a desired one of these information blocks 22 and is operative to combine the mass letter information stored in the selected information block 22 with an address from an information block 13 in the first storage portion 8 upon selection by the keyboard 2. The combined address and mass letter information may then be printed by the printer for dispatch to the desired recipients. This may automatically be done for any selected group of recipients. The information stored in the individual storage blocks 22 of the seventh storage portion 21 may include marked positions to be supplemented by information at correspondingly marked positions in the information blocks 13 of the first storage portion 8. Thereby standard mass letter texts may be caused to include information specific to each recipient in addition to the address.

From the above description it is evident that the system is of high utility for all users which daily dispatch substantial amounts of parcels or other pieces to be shipped. Such users include replacement parts by suppliers, mail-order houses, banking houses, insurance companies, governmental authorities and other service organizations.

Following is an explanation of the program routines which enable the system to carry out functions described above.

FIGS. 3(*a*) and 3(*b*) are a flowchart for the main routine for the system.

The main routine begins when the system is switched on (step 104). The processor 1 then causes to be displayed on the display 5 the date and the time of day in an appropriate mask (step 106). The processor then waits until the user confirms the displayed date and time or enters the new date and time (step 108). Then the processor causes to be displayed the various functions which are available and identifies the keys to be used for selecting the functions (step 110). The processor then waits until a key is depressed (step 112). Once a key has been depressed the processor then determines whether it was the "F1" key that was depressed (step 114). If so, the processor implements the routine which allows entry and storage of recipient information (FIG. 4(*a*)). If not, the processor determines whether the "F2" key was depressed (step 116). If it was the "F2" key, the processor implements the routine for defining a location code (FIG. 5(*a*)). If it was not the "F2" key, the processor determines whether the 37 F3" key was depressed (step 118). If so, the processor implements the routine for defining a print format (FIG. 6(*a*)). If not, the processor determines whether the "F4" key was depressed (step 120). If it was the "F4" key, the processor implements the routine for creating a letter for a mass mailing (FIG. 7(*a*)). If not, the system determines whether the "F5" key was depressed (step 122). If it was the "F5" key, the processor implements the routine for selecting a group of recipients (FIG. 8(*a*)). If not, the processor determines whether the "F6" key was depressed (step 124).

As before, if it was the "F6" key that was depressed, the processor implements the routine for defining a mask (FIG. 9).

If it was not the "F6" key, the processor determines whether the "F7" key was depressed (step 126). If so, the processor implements the routine for storing shipping mode information (FIG. 10). If it was not the "F7" key, the processor determines whether the "F8" key was depressed (step 128). If so, the processor implements the routine for storing rate information (FIG. 11(*a*)). If it was not the "F8" key, the processor determines whether the "F9" key was depressed (step 130). If so, the processor returns to step 110; if not, the system awaits a keyboard entry (step 112).

FIGS. 4(*a*) and (*b*) are a flowchart for storing recipient information. At the beginning of the routine, the processor 1 causes to be displayed on the display 5 a mask for entry of recipient information (step 202). Included in the mask is information identifying the functions of the various function keys. The processor then waits until a key is depressed (step 204). Once a key has been depressed, then the processor determines whether the key depressed was an alphanumeric key (step 206). If so, the processor places the corresponding character in an input buffer (step 208) and then waits for another key to be depressed. If not, the processor determines whether the "F1" was depressed (step 210). If it was the "F1" key, the processor causes to be displayed within the current screen the help mask associated with the portion of the screen mask for which the user is attempting to input information (step 212). The processor then waits for another keystroke. If it was not the "F1" key that was depressed, the processor determines whether it was the "F9" key (step 214). If so, the processor clears the contents of all fields within the current screen mask (step 216) and then waits for another keystroke.

If it was not the "F9" key that was depressed, the processor determines whether it was the "enter" key (step 218). If so, the processor enters the character string which is in the input buffer into the field or page of the screen mask which the user has been attempting to complete (step 220). The processor then waits for another keystroke.

If it was not the "enter" key that was depressed, the processor determines whether it was the "F10" key (step 222). If so, the processor causes to be stored in an information block 13 of storage portion 8 the data which is contained in the fields of the screen mask (step 224). The processor then waits for another keystroke.

If it was not the "F10" key that was depressed, the processor determines whether it was the "F5" key (step 226). If so, the processor causes to be displayed the next page of the screen mask for entry of supplementary information (step 228). The processor then waits for another keystroke..

If it was not the "F5" key that was depressed, the processor determines whether it was the "F4" key (step 230). If so, the processor activates the field of the screen mask for entry of group definition data (step 232). Then the processor waits for another keystroke.

If it was not the "F4" key that was depressed, the processor determines whether it was the "F2" key (step 234). If so, the processor goes to the routine for creating a letter for mass mailing (step 236), as shown on FIGS. 7(*a*) and (*b*). If not, the processor determines whether it was "F6" key that was depressed (step 238). If it was the "F6" key, the processor clears all fields of the screen mask (step 240) and waits for another keystroke.

If it was not the "F6" key that was depressed, the processor determines whether it was the 37 F3" key (step 242). If so, the processor procedes to obtain the required shipping mode and rate information and print required shipping documents as illustrated on FIGS. 12(*a*) and (*b*) (step 244).

If it was not the "F3" key that was depressed, the processor determines whether it was the "F7" key (step 246). If so, the processor goes to the routine for defining printing formats (step 248) as shown in FIGS. 6(*a*) and (*b*). If it was not the "F7" key, the processor determines whether the "F8" was depressed (step 250). If so, the processor causes to be printed an appropriate shipping document using the address information currently displayed (step 252). If not, the processor determines whether it was the "ESC" (i.e. "escape") key that was depressed (step 254). If so, the processor returns to the main routine. If not, the processor waits for another keystroke.

FIGS. 5(*a*) and (*b*) show the routine that allows the user to define location codes. The routine begins with step 302 in which an appropriate mask is displayed, including an identification of the functions performed by the various function keys. The processor then waits for a key to be depressed (step 304). Then the processor determines whether the depressed key was an alphanumeric key (step 306). If so, the processor places the character in an input buffer (step 308) and then waits for a keystroke.

If it was not an alphanumeric key, the processor determines if it was the "F1" key (step 310). If so, the processor displays within the current screen mask a help mask associated with the field into which the user is attempting to enter data (step 312). Then the processor waits for another keystroke.

If it was not the "F1" key, the processor determines if it was the "F9" key (step 314). If so, the processor clears the contents of all fields of the current screen masks (step 316) and waits for another keystroke.

If it was not the "F9" key, the processor determines if it was the first column of the mask which was activated (step 318). If so, the processor causes the character string that is in the input buffer to be entered as a user defined code (step 320) then the processor waits for another keystroke.

If the first column was not activated, the processor determines whether the second column was activated (step 322). If so, the processor causes the character string that is in the input buffer to be entered as the name of the city of the recipient's address (step 324). Then the processor waits for another keystroke.

If the second column was not activated, the processor determines if the third column was not activated (step 326). If so, the processor causes the character string that was in the input buffer to be entered as the postal or zip code of the recipient's address (step 328). Then the processor waits for another keystroke.

If the third column was not activated, the processor determines if the "F10" key was depressed (step 330). If so, the processor causes the contents of the first, second and third columns of each line of the screen mask to be stored in the conversion table (step 332). Then the processor waits for another keystroke.

If it was not the "F10" key that was depressed, the processor determines whether it was the "F6" key (step 334). If so, the processor causes to be erased from the conversion table the line of the screen mask which was currently activated (step 336). Then the processor waits for another keystroke.

If it was not the "F6" key that was depressed, the processor determines if it was the "F3" key (step 338). If so, the processor locates a line in the conversion table based upon the character string that is in the input buffer and the line of the screen.

If it was not the "F3" key that was depressed, the processor determines whether it was the "ESC" (step 342). If so, the processor returns to the main routine. If not, processor waits for another keystroke.

FIGS. 6(*a*) and (*b*) show a flowchart for the routine which allows the user to define a printing format.

This routine begins with display of an appropriate screen mask, which includes identification of the functions performed for the various function keys (step 402). Then the processor waits for a key to be depressed (step 404).

Once a key has been depressed, processor determines whether it was an alphanumeric key (step 406). If so, the processor places the appropriate character in an input buffer (step 408) and then waits for another keystroke.

If it was not an alphanumeric key that was depressed, the processor determines if it was the "F1" key (step 410). If so, the processor causes to be displayed within the current screen mask a help mask associated with the item in the screen which the user is attempting to complete (step 412). Then the processor waits for another keystroke.

If it was not the "F1" key that was depressed, the processor determines whether it was the "F9" key (step 414). If so, the processor clears the contents of all fields of the current screen mask (step 416) and then waits for another keystroke.

If it was not the "F9" key, the processor determines if it was the "enter" key (step 418). If so, the processor causes the character string that is in the input buffer to be inserted into the currently activated field or page of the screen mask and then activates the next field or page (step 420). Then the processor waits for another keystroke.

If it was not the "enter" key that was depressed, the processor determines if it was the "F10" key (step 422). If so, the processor causes the information displayed within the screen mask to be stored in an information block 16 within the fourth storage portion 11 (step 424). Then the processor waits for another keystroke.

If it was not the "F10" key that was depressed, the processor determines if it was the "F2" key (step 426). If so, the processor causes to be displayed the next screen mask which contains information for defining the lines and columns of the printing fields for a particular form (step 428). The processor then waits for another keystroke.

If it was not the "F2" key that was depressed, the processor determines if it was the "F4" key (step 430). If so, the processor deletes the currently displayed form (step 432) and then waits for another keystroke.

If it was not the "F4" key, the processor determines if it was the "ESC" (step 434). If so, the processor returns to the main routine. If not, the processor waits for another keystroke.

FIGS. 7(*a*) and (*b*) show a flowchart for a routine which allows the user to create a letter for a mass mailing.

The routine begins with step 502, in which an appropriate mask is displayed. Then the processor waits for the user to use enter via the keyboard the number of the printer to be used (step 504) then a mask is displayed that allows the user to create the text for the letter (step 506). Thereafter the processor waits for a key to be depressed (step 508).

Once a key has been depressed, the processor determines if it was the "F1" key (step 510). If so, the processor causes to be displayed an appropriate help information mask (step 512) and then waits for another key to be depressed.

If it was not the "F1" key that was depressed, the processor determines if it was an alphanumeric key (step 514). If so, the processor places the character associated with that key in the input buffer (step 516) and then waits for another key to be depressed.

If it was not an alphanumeric key, the processor determines if it was one of the editing keys (step 518). If so, the text in the input buffer is edited in the manner appropriate to the particular editing key that was depressed (step 520). Then the processor waits for another key to be depressed.

If it was not one of the editing keys, the processor determines if it was the "ESC" key (step 522). If not, the processor waits for another keystroke. If it was the "ESC" key, the processor prompts the user to enter a name for the letter that has just been created (step 524). Then the processor stores the letter under that name (step 526) and returns to the main routine.

FIGS. 8 (*a*) and (*b*) show a flowchart for a routine that allows the user to select a group of recipients from among those for whom information has been stored in the memory.

The routine begins with step 602, in which an appropriate mask is displayed, including identification of the functions performed by the various function keys. The processor then waits for a key to be depressed (step 604).

Once a key has been depressed, the processor determines if it was an alphanumeric key (step 606). If so, the processor places in an input buffer the character associated with that key (step 608), and then waits for the next keystroke.

If it was not an alphanumeric key, the processor determines if it was the "F1" key (step 610). If so, the processor causes to be displayed help information relating to the field in the screen mask for which the user was attempting to input data (step 612). Then the processor waits for the next keystroke.

If it was not the "F1" key that was depressed, the processor determines if it was the "F5" key (step 614). If so, the processor displays another screen mask which includes a recipient's address and then allows the user to define group selection criteria through the keyboard (step 616).

If it was not the "F5" key, the processor determines if it was the "F6" key (step 618). If so, the processor causes to be displayed a mask which provides information regarding the possible output modes and allows the user to select an output mode (step 620).

If it was not the "F6" key, the processor determines if it was the "F7" key (step 622). If so, the processor causes to be displayed a mask containing information on possible selection sequences and then allows the user to choose a selection sequence (step 624).

If it was not the "F7" key, the processor determines if it was the "F8" key (step 626). If so, the processor selects the group of recipients and outputs that group in accordance with the choices previously made by the user (step 628).

If it was not the "F8" key, the processor determines if it was the "return" key (step 630). If so, the processor returns to the main routine, if not, the processor waits for another keystroke.

FIG. 9 is a flowchart for the routine which allows the user to customize screen masks.

This routine begins with step 702 in which an appropriate screen mask is displayed, including identification of the functions performed by the various function keys. Then the processor waits for a key to be depressed (step 704).

Once a key has been depressed the processor determines if it is an alphanumeric key (step 706). If so, the processor places in an input buffer the character associated with that key (step 708) and then waits for the next key to be depressed.

If it was not an alphanumeric key, the processor determines if it was the "F1" key (step 710). If so, the processor causes to be displayed within the current screen mask a help information mask associated with the portion of the current screen mask into which the user was attempting to input data (step 712). Then the processor waits for another keystroke.

If it was not the "F1" key, the processor determines if it was the "F2" key (step 714).

If so, the processor uses the character string that is in the input buffer as the designation of the currently activated field of the screen mask and then the processor activates the next field (step 716). Then the processor waits for the next key stroke.

If it was not the "F2" key, the processor determines if it was the "F10" key (step 718). If so, the processor causes to be stored in memory the screen mask as it is currently displayed (step 720). Then the processor waits for the next keystroke.

If it was not the "F10" key, the processor determines if it was the "ESC" key (step 722) if so, the processor returns to the main routine. If not, the processor waits for the next keystroke.

FIG. 10 shows a flowchart for a routine which allows the user to store shipping mode information.

The routine begins with step 802, in which an appropriate screen mask is displayed, including identification of the functions performed by the various function sheets. Then the processor waits for a key to be depressed (step 804).

Once a key has been depressed, the processor determines if it was an alphanumeric key (step 806). If so, the processor places in an input buffer the character associated with that key (step 808) and then waits for the next keystroke.

If it was not an alphanumeric key, the processor determines if it was the "F1" key (step 810). If so, the processor causes to be displayed within the current screen mask a help information mask associated with the field into which the user is attempting to enter data (step 812). Then the processor waits for the next keystroke.

If it was not "F1" key, the processor determines if it was the "F9" key (step 814). If so, the processor clears the contents of all fields of the screen mask (step 816). Then the processor waits for the next keystroke.

If it was not the "F9" key, the processor determines if it was the "enter" key (step 818). If so, the processor causes to be entered into the currently activated field of the screen mask the character string that is stored in the input buffer; the processor then activates the next field (step 820). This step may also include the rate shopping function described above.

If it was not the "enter" key that was depressed, the processor determines if it was the "F10" key (step 822). If so, the processor causes the information contained in the fields of the screen mask to be stored in an information block 14 in the second storage portion 9 (step 824). Then the processor waits for another keystroke.

If it was not the "F10" key, the processor determines if it was the "ESC" key (step 826). If so, the processor returns to the main routine. If not, the processor waits for the next keystroke.

FIGS. 11(*a*) and (*b*) show a flowchart for a routine which allows the user to store rate information. The routine begins with step 902, in which an appropriate screen mask is displayed, including identification of the functions performed by the various function keys. Then the processor waits for a key to be depressed (step 904).

Once a key has been depressed, the processor determines whether it was an alphanumeric key (step 906). If so, the processor places in an input buffer the character associated with that key (step 908) and then it waits for another keystroke.

If it was not an alphanumeric key, the processor determines whether it was the "F1" key (step 910). If so, the processor causes to be displayed within the current screen mask an additional help information mask associated with field into which the user was attempting to input data (step 912). Then the processor waits for the next keystroke.

If it was not the "F1" key, the processor determines if it was the "F2" key (step 914). If so, the processor allows entry of zone and postal code information in tabular form (step 916).

If it was not the "F2" key, the processor determines if it was the "F3" key (step 918). If so, the processor allows entry in a zone table of weight values and associated rate values (step 920).

If it was not the "F3" key, then the processor determines if it was the "F4" key (step 922). If so, the processor allows entry of distance, weight and associated rate values for a distance related rate table (step 924).

If it was not the "F4" key, the processor determines if it was the "F7" key (step 926). If so, the processor causes the table information to be stored in an information block 15 of the third storage portion 10 (step 928).

If it was not the "F7" key, the processor determines if it was the "ESC" key (step 930). If so, the processor returns to the main routine. If not, the processor waits for the next keystroke.

We claim:

1. An apparatus for preparing shipping documents required by parcel carriers who have different types of forms for shipment of articles, comprising:

(a) weight data entry means for entering data representing a weight of an article to be shipped, a printer, a memory, a keyboard;

(b) a processor connected to the weight data entry means, the printer, the memory, and the keyboard;

(c) first means for feeding single forms to the printer;
(d) second means for storing a batch of forms and feeding the forms of the batch to the printer; and
(e) third feeding means for feeding continuous forms to the printer;

said memory storing carrier form data for a plurality of carriers and said keyboard being operable to send a carrier selection signal, said signal indicating selection of one of said plurality of carriers; and
said processor being programmed to:
(i) receive said carrier selection signal; and
(ii) select, in accordance with the carrier form data for said selected one of said plurality of carriers, one of said first, second and third means to feed a form to said printer.

2. The apparatus of claim 1, wherein said first means includes means for feeding forms of different dimensions.

3. The apparatus of claim 2, wherein said second means includes means for storing and feeding forms of different dimensions.

4. The apparatus of claim 1, wherein the weight data entry means is a scale generating an electrical output signal which corresponds to the weight of the article to be shipped.

5. The apparatus of claim 1, wherein the weight data entry means is a light scanning, bar code reader connected to the processor.

6. The apparatus of claim 1, wherein the weight data entry means is the keyboard.

7. The apparatus of claim 1, wherein the processor is a microprocessor.

8. The apparatus of claim 1, wherein:
the stored carrier form data for at least one of said plurality of carriers indicates that said at least one carrier requires at least two forms; and
said at least one carrier having been selected, said processor selects a first one of said first, second and third means to feed a first of said at least two forms and subsequently selects a second one of said first, second and third means to feed a second of said at least two forms.

9. In an apparatus for preparing shipping documents relating to the shipment of parcels, said apparatus having weight data entry means, a printer, a memory, a keyboard and a processor connected to the weight data entry means, the printer, the memory and the keyboard, the improvement comprising:
(a) means for storing in the memory names and addresses of recipients of parcels;
(b) means for storing in the memory second information relative to parcels received by the recipients;
(c) first means for feeding single forms to the printer;
(d) second means for storing a batch of forms and feeding the forms of the batch to the printer; and
(e) third means for feeding continuous forms to the printer;

said memory storing carrier form data for a plurality of carriers; said form data identifying which and how many forms are required by each of said carriers, said memory storing said recipients' preferred carrier of said plurality of carriers, said keyboard being operable to send a recipient selection signal, said recipient selection signal indicating selection of one of said recipients, said processor being programmed to:
(i) receive said recipient selection signal;
(ii) refer to said stored carrier form data for said selected recipient's preferred carrier; and
(iii) sequentially select, in accordance with the form data for said selected recipient's preferred carrier, at least one of said first, second and third means;

whereby said at least one of said first, second and third means feeds to the printer said identified required forms for said preferred carrier upon selection of said selected recipient.

10. The improvement of claim 9, wherein said second information comprises information relative to goods purchased by the recipients.

11. The improvement of claim 10, wherein said information relative to goods purchased includes dates of purchase.

12. The improvement of claim 11, wherein said information relative to goods purchased further includes prices of the goods purchased.

13. The improvement of claim 12, further comprising means associated with the processor for transmitting in computer-receivable form said information relative to the parcels.

14. The improvement of claim 9, wherein said recipient selection signal comprises the name of one of said recipients.

15. The improvement of claim 9, wherein said recipient selection signal consists of a portion of the name of one of said recipients.

16. The improvement of claim 9, wherein said recipient selection signal comprises the address of one of said recipients.

17. The improvement of claim 9, wherein said recipient selection signal consists of a portion of the address of one of said recipients.

18. The improvement of claim 9, wherein said recipient selection signal comprises an identification code that identifies only one of said recipients.

19. The improvement of claim 9, wherein said recipient selection signal consists of a portion of an identification code that identifies only one of said recipients.

20. The improvement of claim 9, further comprising a display connected to the processor and means associated with the keyboard for defining a format in which information relative to said selected recipient is to be displayed.

* * * * *

US005602382A

United States Patent [19]

Ulvr et al.

[11] **Patent Number: 5,602,382**

[45] **Date of Patent: Feb. 11, 1997**

[54] **MAIL PIECE BAR CODE HAVING A DATA CONTENT IDENTIFIER**

[75] Inventors: **Joseph Ulvr**, Carp; **Adrian T. S. C. Kho**, Kanata, both of Canada

[73] Assignee: **Canada Post Corporation**, Ontario, Canada

[21] Appl. No.: **331,955**

[22] Filed: **Oct. 31, 1994**

[51] **Int. Cl.⁶** **G06K 19/06**

[52] **U.S. Cl.** **235/494**; 235/463; 235/462; 235/456; 235/375; 371/37.1

[58] **Field of Search** 235/494, 463, 235/462, 456, 375; 371/37.1, 38.1, 39.1, 40.1; 209/584, 900

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,991,300	11/1976	Chadima, Jr.	235/61.12
4,013,997	3/1977	Treadwell, III	371/37.1
4,742,521	5/1988	Nishida	371/84
4,852,099	7/1989	Ozaki	371/37
4,864,112	9/1989	Imai et al.	235/463
4,874,936	10/1989	Chandler et al.	235/494
4,896,029	1/1990	Chandler et al.	235/494
4,972,475	11/1990	Sant'Anselmo	380/54
4,998,010	3/1991	Chandler et al.	235/494
5,053,609	10/1991	Priddy et al.	235/436
5,068,858	11/1991	Blaum et al.	371/41
5,070,504	12/1991	Bossen et al.	371/54
5,099,484	3/1992	Smelser	371/38.1
5,107,503	4/1992	Riggle et al.	371/37.1

(List continued on next page.)

FOREIGN PATENT DOCUMENTS

984971	3/1976	Canada .
998474	10/1976	Canada .
1003966	1/1977	Canada .
1004767	2/1977	Canada .
1070837	1/1980	Canada .
1120884	3/1982	Canada .
1145847	5/1983	Canada .
1152648	8/1983	Canada .
1173962	9/1984	Canada .
1202929	4/1986	Canada .
1241443	8/1988	Canada .
1264341	1/1990	Canada .
2019956	12/1990	Canada .
2006898	6/1991	Canada .

(List continued on next page.)

OTHER PUBLICATIONS

M. Mansour, "Multi-Tiered Condensed Bar Code", vol. 26, No. 2, Jul. 1983 (IBM Technical Disclosure Bulletin).

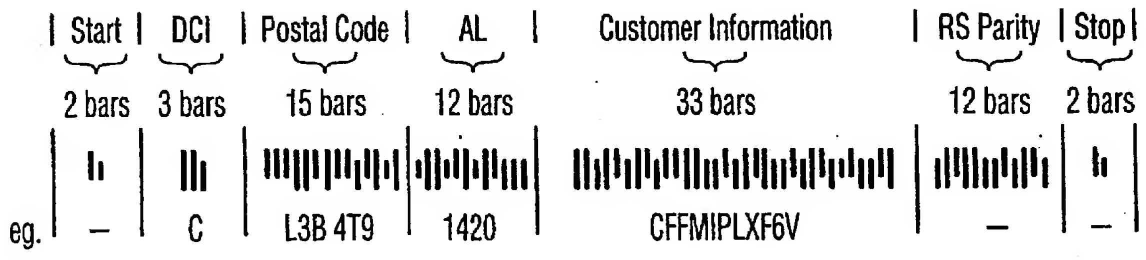
R. E. Blahut, K. Jones, J. A. Fairless, "A Damage Resistant Bar Code for the Royal Mail", pp. 49–54; IBM 156.

Primary Examiner—Donald T. Hajec
Assistant Examiner—Michael G. Lee
Attorney, Agent, or Firm—Wenderoth, Lind & Ponack

[57] **ABSTRACT**

A bar code for mail pieces uses bars each of which has four possible states. Two different bars indicate the start of the code and the same two bars in the same order indicate the end of the code. A data content identifier follows the start bars and this indicates the structure and length of the following data field so that when the code is read it will be recognized and read properly. The use of the data content identifier allows the code to be used for different customer and Post Office applied applications in which the code structure, length and content varies. The data field may contain a postal code with or without an address locator, a machine ID, customer information and service information. The code may include a country code field for mail pieces that are being mailed to a different country. The code may also include a field indicating whether the codeword is complete or whether it has to be concatenated with a preceding or subsequent codeword. Error protection in all cases is provided by a Reed-Solomon parity field following the data field. For customer applied codes this parity field may be made shorter than for Post Office applied codes because the potential for error in printing the code by the customer is less in view of the fact that he has more control over the paper quality, colour, extraneous markings, etc.

27 Claims, 5 Drawing Sheets



U.S. PATENT DOCUMENTS

5,124,536	6/1992	Priddy et al.	235/432
5,126,542	6/1992	Priddy et al.	235/456
5,153,929	10/1992	Itagaki	382/65
5,157,669	10/1992	Yu et al.	371/37.7
5,161,163	11/1992	Bossen et al.	371/54
5,168,509	12/1992	Nakamura et al.	375/39
5,206,490	4/1993	Petigrew et al.	235/462
5,226,932	7/1993	Prasad	55/16
5,235,172	8/1993	Oehlmann	235/494
5,280,488	1/1994	Glover et al.	371/37.1
5,288,976	2/1994	Citron et al.	235/375
5,298,731	3/1994	Ett	235/494
5,304,786	4/1994	Pavlidis et al.	235/462
5,324,927	6/1994	Williams	235/494
5,420,403	5/1995	Allum et al.	235/375
5,475,716	12/1995	Huang	371/37.1 X
5,479,515	12/1995	Longacre, Jr.	235/463 X

FOREIGN PATENT DOCUMENTS

2020475	6/1991	Canada .
2022269	7/1991	Canada .
1291247	10/1991	Canada .
1293806	11/1991	Canada .
1295744	2/1992	Canada .
1305257	7/1992	Canada .
1306807	8/1992	Canada .
1311842	12/1992	Canada .
2063103	9/1993	Canada .
2095508	11/1993	Canada .
2140948	12/1984	United Kingdom .

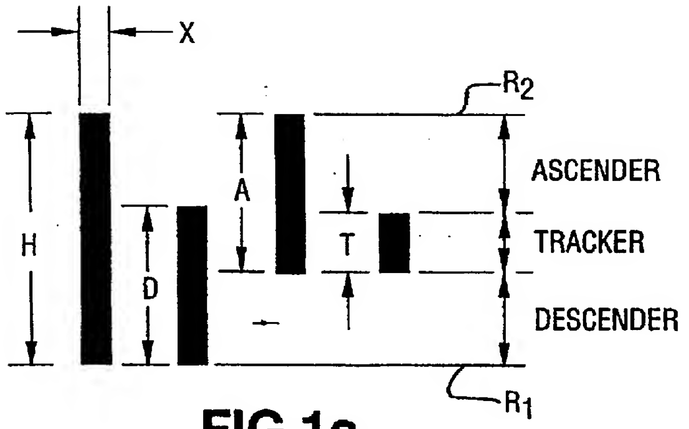

X
R2
H
D
A
T
ASCENDER
TRACKER
DESCENDER
R1

FIG.1a

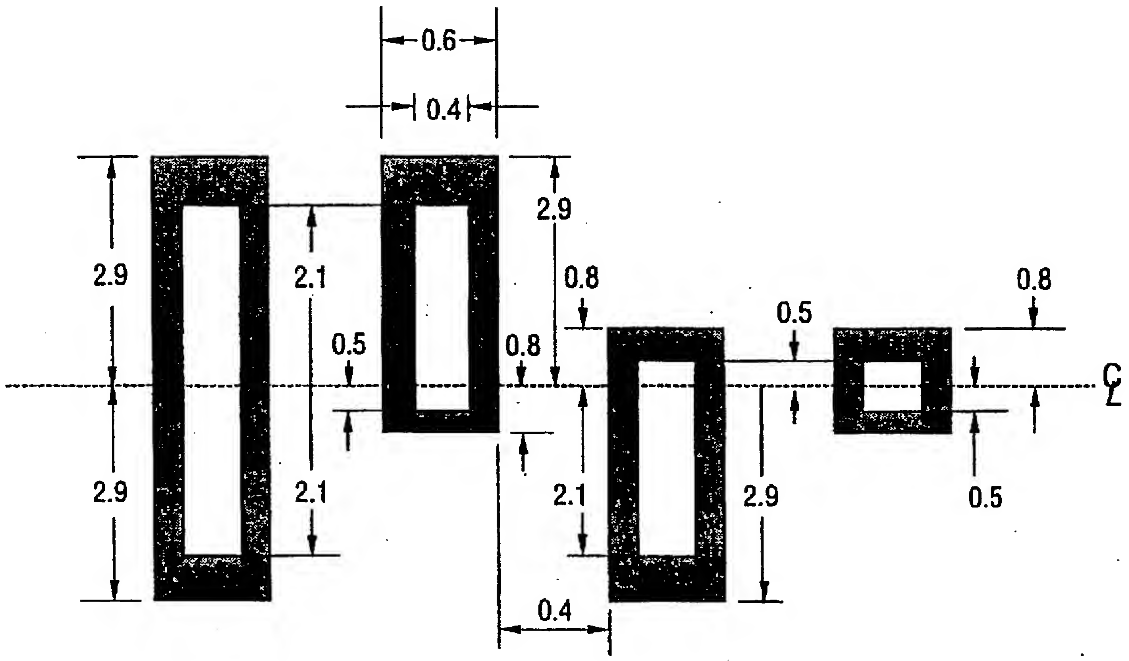

0.6
0.4
2.9
2.1
0.5
0.8
2.9
0.8
0.5
0.8
2.9
2.1
2.1
2.9
0.5
0.4
℄

FIG.1b

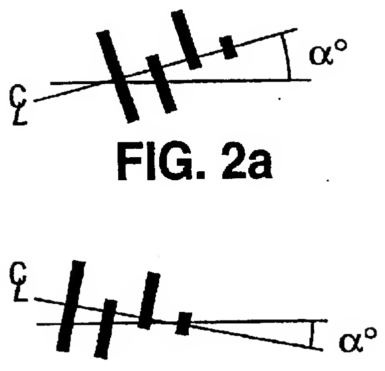


FIG. 2a

FIG. 2b

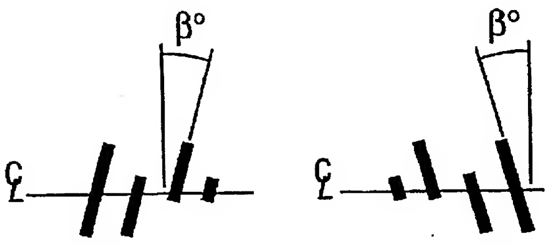


FIG. 2c

FIG. 2d

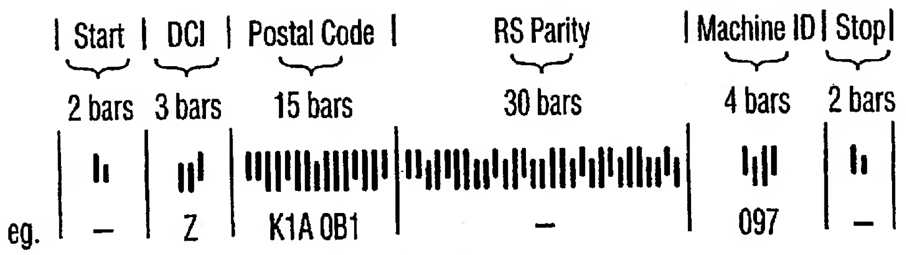


FIG. 3

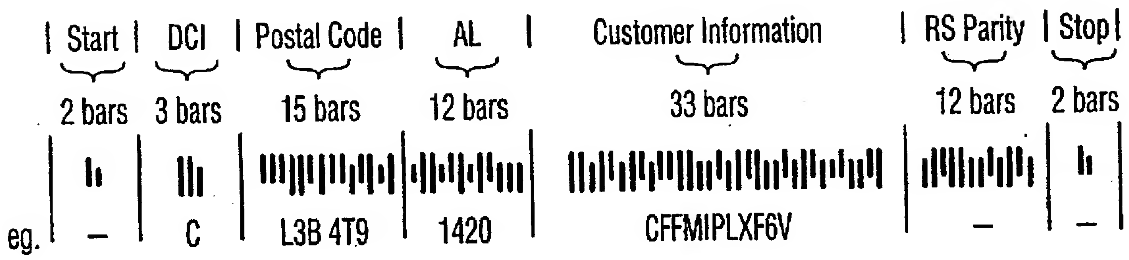


FIG. 4a

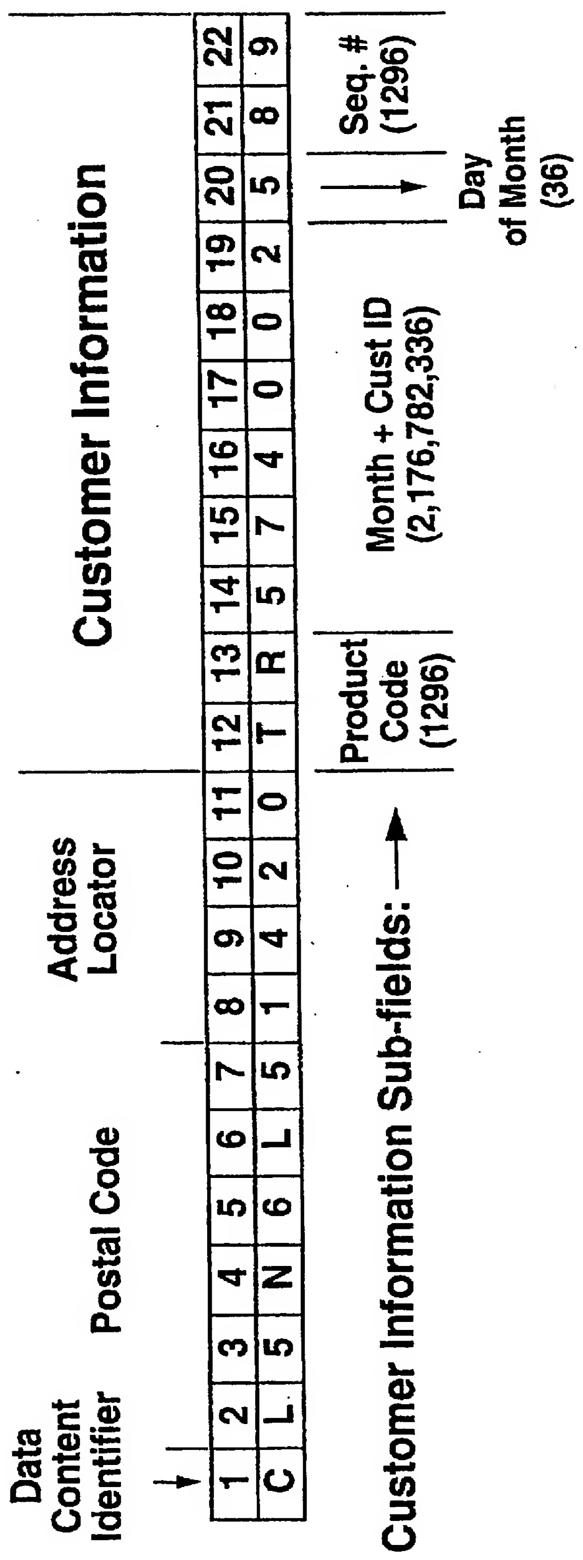

Data Content Identifier
Postal Code
Address Locator
Customer Information
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22
C L 5 N 6 L 5 1 4 2 0 T R 5 7 4 0 0 2 5 8 9
Customer Information Sub-fields:
Product Code (1296)
Month + Cust ID (2,176,782,336)
Day of Month (36)
Seq. # (1296)

FIG. 4b

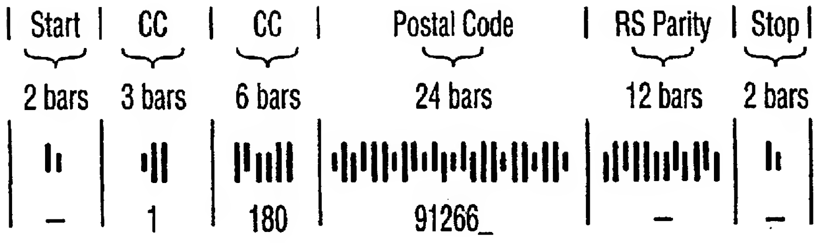


FIG. 5

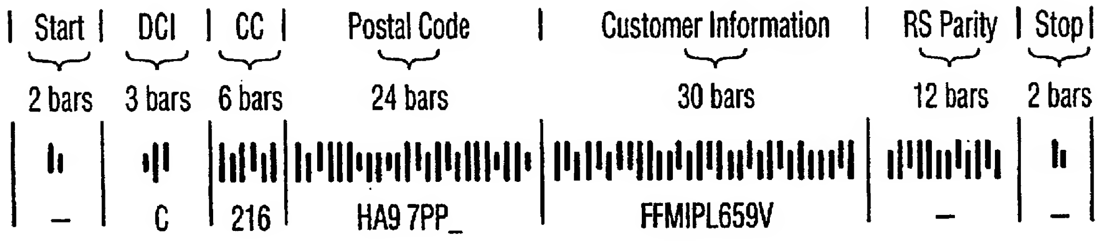


FIG. 11

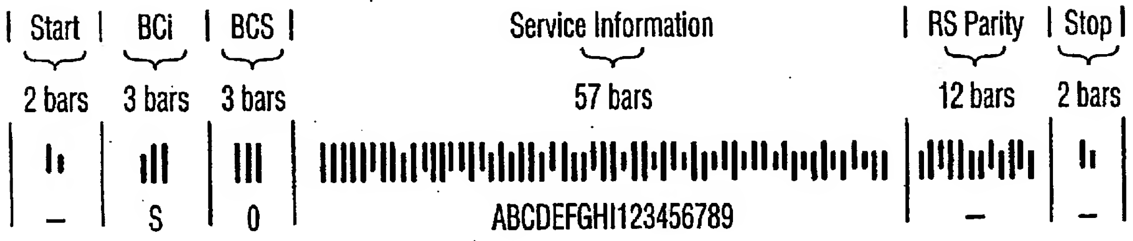


FIG. 6

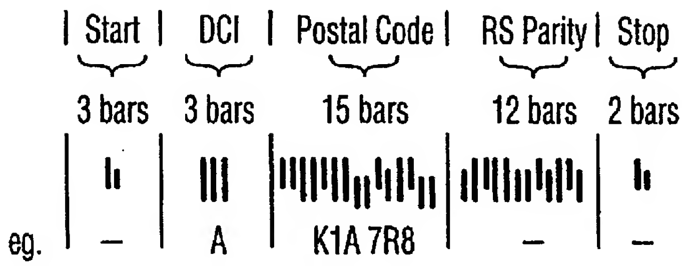


FIG. 7

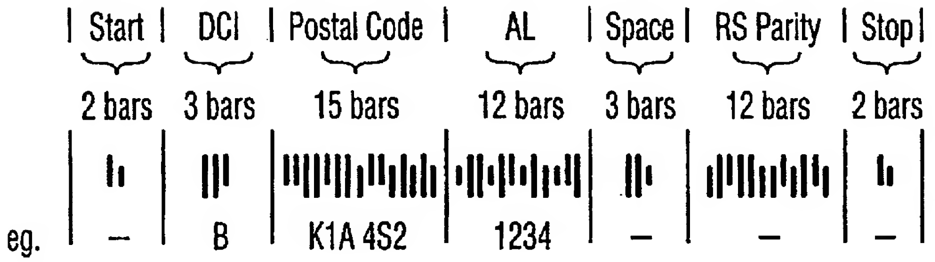


FIG. 8

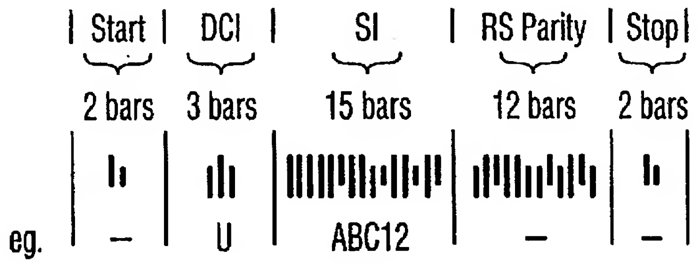


FIG. 9

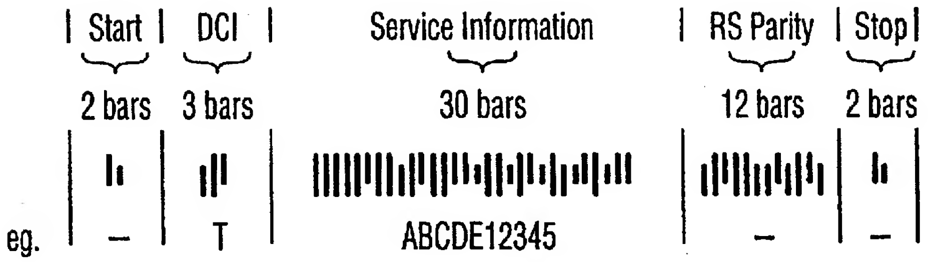


FIG. 10

MAIL PIECE BAR CODE HAVING A DATA CONTENT IDENTIFIER

BACKGROUND OF THE INVENTION

This invention relates to bar codes used in the processing of mail pieces.

In many countries a postal code is used to facilitate automation in sorting. In Canada, the postal code contains three alphabetic characters (letters) interleaved with three numeric characters (numbers) while in the U.S.A. the zip code consists of five numbers. If a customer has applied the postal code to an envelope this is converted by an optical character reader (O.C.R.) and computer in the Post Office to a bar code which is then printed on the envelope. If the customer has not applied the postal code, this will be generated in the Post Office and the bar code will be printed on the envelope as before.

It is also becoming more usual for the large corporate customer to apply the postal code in bar code format. A bar code is used because it is easier to read automatically than alphanumeric characters.

The British Post Office (BPO) has developed a 4 state bar code. The four possible states are one which comprises only a tracker element, one which comprises a tracker element and an ascender element, one which comprises a tracker element and a descender element and one which comprises a tracker element, an ascender element and a descender element. These elements will be described in detail hereinbelow.

The BPO code uses four bars to represent each alphanumeric character but for error protection each character must have two ascenders and two descenders which limits the number of possible combinations to 36. Error detection is dealt with by including a check sum. The BPO code is intended to be printed by customers (mailers) to encode sortation information.

The BPO code uses a single bar to indicate the start of the code and a different single bar to indicate the end of the code. The start/stop bars in the BPO code can easily be confused and can result in decoding of an upside down bar code.

A paper entitled "A Damage Resistant Bar Code for the Royal Mail" by Blahut et al, 1992 discusses an improvement of the BPO bar code which essentially makes the BPO code more robust by adding Reed-Solomon error correction to handle missing bars (erasures) and bar print errors. The encoding is based on codewords of two bars each—a left and right codeword—and the interleaving of the codewords to form a bar code. Blahut has also identified some decoding logic to recover from missing or incorrect bars and overcome the weakness of the start/stop pattern with decoding logic. The Blahut Code is intended to be printed by the Post Office as an internal code and only postal sortation data is encoded.

It is an object of the present invention to improve on the BPO and Blahut codes by offering a robust code which is sufficiently flexible that it may be applied by the Post Office or the customer and which can be used not only for the postal code but for route sequencing, track and tracing, revenue accounting and other customer information as well as customer service information such as return mail management, automated data entry of customer information and the like.

SUMMARY OF THE INVENTION

According to a broad aspect of the invention, a mail piece bears a bar codeword containing information for the pro-

cessing of the mail piece, the bar codeword having a plurality of parallel bars each of which has a state selected from a plurality of possible states. The bar codeword comprises a start field followed by a data content identifier (DCI) field specifying the structure of the codeword followed by at least one data field, followed by a Reed-Solomon parity field, followed by a stop field.

In one embodiment of the invention, as in Blahut, bars which have four possible states are used but the invention is not limited to the use of a four state code.

The data field may contain different types of information depending on the application. For example, it may contain a postal code in the ANANAN format where A is an alphabetical character and N is a numerical character. The A's are each represented by three bars and the N's are each represented by two bars.

The use of the start and stop bars provides an indication of correct orientation and direction of reading in a single efficient manner.

Not only postal (sortation and sequencing) data can be encoded but also customer data to support the creation of value added products and services selectable by the customer at the time of printing the mail pieces.

The new coding is space efficient because in a preferred embodiment it encodes characters in 3 bars with the numerics in the CPC postal code using only 2 bars each.

The code is highly damage resistant and provides varying levels of error correction appropriate to the application. For example, more error correction is provided in the internal applied codes than the customer applied codes.

The invention also enables in a specific embodiment the concatenation of multiple bar codes.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1*a* illustrates the basic elements of the four state code used in a preferred embodiment of the invention;

FIG. 1*b* illustrates the minimum and maximum dimensions of the bars used in the four state code;

FIGS. 2*a* to 2*d* illustrate how the code pattern or the individual bars within the code may be skewed;

FIG. 3 illustrates a typical bar code employing the invention;

FIG. 4*a* illustrates another example of a bar code employing the invention;

FIG. 4*b* illustrates a bar code similar to that of FIG. 4*a* in which the customer field has been broken down into subfields; and

FIGS. 5 through 11 illustrate further examples of bar codes employing the invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. 1, the basic elements of the printed four state code are four vertical bars which differ in length and/or starting point with respect to the horizontal.

More particularly, there is a full height bar H which is the longest of the four bars and extends between lower and upper horizontal references R_1 and R_2, respectively. The bar immediately to the right of bar H is a bar D which is greater than half of the height of bar H and extends from above the mid-point of bar H down to the level of reference R_1. Immediately to the right of bar D is a bar A which has the same height as bar D but extends from below the mid-point of bar H up to the level of reference R_2. The final element is a bar T which is centred about the mid-point of bar H and which has a height represented by the overlap of bars A and D.

Another way of defining bars H, D, A and T is in terms of the three basic elements, Tracker, Ascender and Descender shown in FIG. *1a*. The Tracker element is a short element centred exactly between the lower and upper references R_1 and R_2. The Ascender and Descender elements are identical in length, the Ascender extending upwardly from the upper limit of the Tracker to reference R_2 and the Descender extending downwardly from the lower limit of the Tracker to reference R_1.

The Tracker is present in all of the four bars. In the T bar the Tracker is the only element, the D bar consists of Tracker and Descender, the A bar consists of Tracker and Ascender and the H bar consists of the Tracker, Ascender and Descender. The four possible bars and their assigned numerical values can be summarized as follows:

BAR	ELEMENTS	VALUE
T	Tracker	3
D	Tracker and Descender	2
A	Tracker and Ascender	1
H	Tracker, Ascender and Descender	0

The maximum and minimum permissible dimensions of elements A, D, H, T and bar width X are indicated in FIG. *1b* where the maximum bar outlines are the shaded portions, and are summarized below:

	Minimum		Maximum	
Element	mm	in.	mm	in.
T	1.0	0.04	1.6	0.06
A	2.6	0.10	3.7	0.145
D	2.6	0.10	3.7	0.145
H	4.2	0.165	5.8	0.23
X (bar width)	0.4	0.015	0.6	0.025
bar gap	0.4	0.015		

Note: The minimum gap between bars takes priority over all other dimensions.

The bar density should be around 20–24 bars per 25.4 mm. FIG. 2 illustrates two types of skew that can occur when printing bar codes. FIGS. *2a* and *2b* illustrate code skew α in which the entire code pattern is skewed with respect to the bottom edge of the mail piece and FIGS. *2c* and *2d* illustrate bar skew β in which individual bars are skewed with respect to the centreline of the code pattern.

For code skew the acceptable limit is less than ±5° from the horizontal and for bar skew the limit is less than ±5° from the vertical. It is possible for both types of skew to occur on a single item and in that case the total skew $|\alpha|+|\beta|$ should be less than 5°.

Canada Post Corporation (CPC) proposes using the basic four state bar code in different applications. The term PostBar has been coined by CPC to refer to the basic four state bar code and the letters xyz appear after the designation PostBar where

"x" is

"D" for domestic (Canada) applications

"G" for global (international) applications

"C" for CPC internal applications and

"S" for service applications.

"yz" specifies the number of characters in the bar code.

The PostBar applications are as follows:

Domestic	Global	Service	Internal
PostBar.D07	PostBar.G12	PostBar.S06	PostBar.C10
PostBar.D12	PostBar.G22	PostBar.S11	
PostBar.D22		PostBar.S21	

Let us consider in detail PostBar.C10 to illustrate how the four state code is applied. The format of this code is illustrated in FIG. 3. This may be summarized as follows:

DATA FIELD	BARS	DATA CHARACTERS
Start/synchronization	2	
Data Content Identifier (DCI)	3	Z
Postal code (FSA LDU)	15	ANANAN
RS Parity Check	30	
Machine ID	4	BBBB
Stop/synchronization	2	
Total	56	

The data characters are denoted A, N, Z and B. A is an alphabetic character and is denoted by 3 bars, N is a numeric character and is denoted by 2 bars, Z is an alphanumeric (i.e., either alphabetic or numeric) character and is denoted by 3 bars and B is 1 bar.

Table 1 shows the encoding for the "A" and "N" characters and Table 2 shows the encoding for the "Z" characters.

TABLE 1

'A' CHARACTERS		'N' CHARACTERS	
Letter	Bars	Number	Bars
A	HHD	0	HH
B	HAH	1	HA
C	HAA	2	HD
D	HAD	3	AH
E	HDH	4	AA
F	HDA	5	AD
G	HDD	6	DH
H	HHA	7	DA
I	AHA	8	DD
J	AHD	9	TH
K	AAH		
L	AAA		
M	HHH		
N	ADH		

TABLE 1-continued

'A' CHARACTERS		'N' CHARACTERS	
Letter	Bars	Number	Bars
O	ADA		
P	ADD		
Q	DHH		
R	DHA		
S	DHD		
T	DAH		
U	DAA		
V	DAD		
W	DDH		
X	DDA		
Y	DDD		
Z	AHH		

TABLE 2

'Z' CHARACTERS	
Symbol	Bars
Space	HHT
A	HHH
B	HHA
C	HHD
D	HAH
E	HAA
F	HAD
G	HDH
H	HDA
I	HDD
J	AHH

TABLE 2-continued

'Z' CHARACTERS	
Symbol	Bars
K	AHA
L	AHD
M	AAH
N	AAA
O	AAD
P	ADH
Q	ADA
R	ADD
S	DHH
T	DHA
U	DHD
V	DAH
W	DAA
X	DAD
Y	DDH
Z	DDA
0	DDD
1	THH
2	THA
3	THD
4	TAH
5	TAA
6	TAD
7	TDH
8	TDA
9	TDD

Note that the encoding shown in Table 1 is used only for the postal code mapping. As seen in Table 1 only the 9 digit contains the tracking element T which because of its size is the most likely of the four elements to be obscured or missing. This minimization of the occurrence of the T bars provides extra security for the Postal Code.

With regard to the "B" bars, these are used only for the Machine ID and can be decoded using a quadral representation with 'n' as the number of bars in the data block and 'V_n' as the value of each bar in the following equation:

$$B_nB_{n-1} \ldots B_1 = V_n x 4^{n-1} + V_{n-1} x 4^{n-2} + \ldots + V_1 x 4^0$$

The bar values (V_n) are assigned as follows: H=0; A=1; D=2; T=3

$$\text{e.g. } ADHA = 1\times4^3 + 2\times4^2 + 0\times4 + 1\times4^0 = 64 + 32 + 0 + 1 = 097$$

For 4 bars, the maximum value is:

$$TTTT = 3\times64 + 3\times16 + 3\times4 + 3\times1 = 225$$

Referring to the various data fields shown in FIG. 3, the first field is START which comprises an A bar followed by a T bar. The last field is STOP which also comprises an A bar followed by a T bar. This sequence provides all orientation or direction of flow of the code so that an upside down label or letter inserted backwards can be identified immediately. The sequence also provides an additional marker for synchronization and a unique identifier so that the code can be recognized immediately.

The next field is the DCI (Data Content Identifier) which specifies the structure and the number of data elements within the bar code. When a bar code reader decodes a DCI it will know how to decode the remaining data elements. The DCI can be either an alphabetic or numeric character ("Z" character) encoded using three bars according to Table 2. Within CPC the DCI's are assigned in the following way:

1–9 Reserved for global (international) applications

A–L Reserved for domestic (Canada) applications

M–U Reserved for service applications

V–Z Reserved for internal applications.

The DCI illustrated in FIG. 3 comprises two D bars followed by one A bar and from Table 2 this corresponds to the letter Z. When the DCI is a Z this specifies that there are 6 decodable characters in the form ANANAN for sortation and a binary machine ID in the form BBBB.

This does in fact correspond with FIG. 3 where the next field is the postal code which in Canada is constructed as alternate alphabetic/numerical characters ANANAN with each letter being formed by 3 bars and each number formed of 2 bars as shown in Table 1. The postal code thus consists of 15 bars. By consulting Table 1, it can be seen the postal code in the example illustrated in FIG. 3 is K1A 0B1.

The next field is the Reed-Solomon Parity Check consisting of 30 bars comprising 10 alphanumeric characters Z. The RS code chosen is a (16,6) Reed-Solomon code over GF(64) which can correct 5 symbol errors and up to 10 symbol erasures (30 bars). That is, more than half the bar code could be missing and the remaining bar code would still be successfully decoded. The error correcting capability of this RS code will be discussed in greater detail below.

The next field is the Machine ID field which identifies the particular machine which applied the bar code. The four bars shown in this example are:

$$ADHA = 1\times4^3 + 2\times4^2 + 0\times4^1 + 1\times4^0 = 64 + 32 + 0 + 1 = 097$$

Turning now to FIG. 4*a*, this illustrates the format of PostBar.D22 which uses a (25,21) Reed-Solomon code over GF(64). PostBar.D22 is a customer applied bar code for domestic Canadian applications. The data structure may be summarized as follows:

DATA FIELD	BARS	DATA CHARACTERS
Start/synchronization	2	
Data Content Indentifier (DCI)	3	Z
Postal code (FSA LDU)	15	ANANAN
Address Locator (AL)	12	ZZZZ
Customer Information	33	ZZZZZZZZZZZ
RS Parity Check	12	
Stop/synchronization	2	
Total	79	

As for the PostBar.C10 code discussed above, data character A is an alphabetic character denoted by 3 bars, N is a numeric character denoted by 2 bars and Z is an alphanumeric character denoted by 3 bars. Table 1 shows the encoding for the "A" and "N" characters and Table 2 shows the encoding for the "Z" characters.

Referring to the various data fields shown in FIG. 4*a*, the start and stop fields, the DCI field and the Postal code fields are identical to the corresponding fields in the PostBar.C10 code. In the particular example shown, by consulting Table 2 it will be seen the DCI corresponds to the letter C and by consulting Table 1 it will be seen the Postal code corresponds to L3B 4T9.

When the DCI is a C this specifies that there are 21 decodable characters which follow in the form of the postal code (ANANAN), address locator (ZZZZ) and 11 customer data characters (ZZZZZZZZZZZ).

Unlike PostBar.C10, PostBar.D22 does not have a Machine ID field as the printer applying the code is not a CPC (Canada Post Corporation) machine and is, therefore, of no real interest.

PostBar.D22 has two fields, namely AL and Customer Information, not present in PostBar.C10. The field AL is an address locator field which consists of 12 bars and appears immediately to the right of the Postal Code. The Customer Information field follows and this has 33 bars. These fields are encoded according to alphanumeric Table 2 and so there are 4 characters for the AL field and 11 for the Customer Information field.

Reference should be made to U.S. patent application Ser. No. 888,905 filed on May 26, 1992 and assigned to Canada Post Corporation, which application is incorporated hereby by reference, for a further explanation of the AL and Customer Information fields. More particularly and in brief, the AL field, referred to in the earlier application as PODI (Point of Delivery Indicator) is a suffix to the postal code which is determined from the address on the mail piece as well as the Postal code. The postal code together with the AL allows a mail piece to be sorted for delivery to the specific address. The term POCI (Point of Call Identifier) has been coined for the combination (Postal Code+the Address Locator).

It is noted that the RS Parity field in PostBar.D22 consists of only 12 bars in contrast to the 30 bars of PostBar.C10. This is because more protection is needed for Post Bar.C10 than for PostBar.D22. This results from the fact that the

PostBar.C10 is a code printed by CPC on mail pieces which have a great variety of surfaces and background and so there is a likelihood of background noise from extraneous printing or marking. On the other hand PostBar.D22 is applied by the customer who has greater control over the printing surface and so there is less potential for background noise.

From a consideration of Table 2 it can be seen that, for the specific example shown for PostBar.D22, the AL is 1420, and the Customer Information is CFFMIPLXF6V. From a consideration of Table 1 the postal code converts to L3B 4T9.

FIG. 4*b* shows how the Customer Information field of FIG. 4*a* may be broken down into sub-fields. In FIG. 4*b* the bars are not shown. The customer data can be broken into sub-fields such as those shown. The Product Code identifies the specific customer, the Sequence Number identifies the batch mailed on a particular day and the Month and Day of Month are self-explanatory. This information uniquely identifies a mail piece and allows track and trace of the mail piece as well as revenue accounting for example.

The numbers in the brackets represent the number of combinations possible for each sub-field.

FIG. 5 illustrates an example of an International or Global code, PostBar.G12 which would be used when the mail piece is addressed to another country. This is a (15,11) Reed-Solomon code over GF(64). The format may be summarized as follows:

DATA FIELD	BARS	DATA CHARACTERS
Start/synchronization	2	
Data Content Identifier (DCI)	3	Z
Country Code (CC)	6	NNN
Postal code*	24	ZZZZZZZZ
RS Parity Check	12	
Stop/synchronization	2	
Total	49	

*Unused characters in the postal code field will be filled with space characters.

The DCI is determined from Table 2 to be 1 and this specifies that there are 11 decodable characters that follow in the form of a three numeric character country code (NNN) and an 8 character postal code (ZZZZZZZZ).

It is noted that in this code no A or N characters of Table 1 are used for the Postal Code. Also, a new field, namely Country Code, is present and as can be seen by consulting Table 1, for the specific example shown this translates to 180 which may, for example, identify the USA. The postal code or zip code is determined from Table 2 to be 91266 followed by three spaces. For the U.S.A. only 15 bars are needed for the ZIP code but the field is provided with 24 bars because other countries require more than 5 characters for the postal code.

FIG. 6 illustrates an example of a customer applied service code, PostBar.S21 which is a (25, 21) Reed-Solomon code over GF(64) having a data structure summarized as follows:

DATA FIELD	BARS	DATA CHARACTERS
Start/synchronization	2	
Data Content Identifier (DCI)	3	Z
Bar Code Sequencer (BCS)	3	Z
Service Information (SI)	57	ZZZZZZZZZZZZZZZZZZZ
RS Parity Check	12	
Stop/synchronization	2	
Total	79	

As for PostBar.G12 discussed above all the data characters are Z characters obtained from Table 2. The DCI is determined from Table 2 to be S and this specifies that there follows as data a 19 character (57 bars) Service Information field which in this case translates to ABCDEFGHIJ123456789 from Table 2. There is no routing data field because this code is used for special services required by a customer. For example the Service Information field could be used for return mail management or to provide other information useful to the customer.

Between the DCI field and the Service Information field is a Bar Code Sequencer (BCS) field which is a single character consisting of 3 bars that allows the concatenation of two bar codes to encode data longer than 19 characters. When a single 19 character code is used the BCS is DDD which from Table 2 is 0. To indicate the first of two concatenated bar codes the BCS would be chosen to be 1 and to indicate the second of two concatenated bar codes the BCS would be chosen to equal 2.

The operation of the error correcting code (ECC) for PostBar.C10 (FIG. 3) and for PostBar.D22 (FIG. 4*a*) will now be discussed in greater detail.

The error correcting code (ECC) for PostBar.C10 protects the postal code and DCI but not the machine ID. The ECC is a (16, 6) Reed-Solomon code defined over the field GF(64) with 64 elements. Precisely, the defining roots of the code are α^i for i=1, 2, . . . , 10 where α is a root of X^6+X+1. Each codeword consists of 6 message (or information) symbols and 10 check symbols. Each symbol is an element of GF(64) and is represented by 3 bars. The ECC can correct up to t symbols with errors and e erased symbols as long as $2t+e\leq 10$. For example, since each symbol corresponds to 3 bars, an erasure of 30 consecutive bars is correctable. One should be aware that if there are e=10 erased symbols then there is no way for the code to detect any additional errors. Any other error in addition to these 10 will force a decoding error. (The constant erase_max in the software can be set less than 10 to stay away from this possibility.)

The error correcting code for PostBar.D22 covers all of the fields except the start and stop bars. The code is a (25, 21) Reed-Solomon code defined over GF(64). It can correct t errors and e erasures in the symbols of the code as long as $2t+e\leq 4$. As before, each code symbol corresponds to 3 bars. The code uses 4 check symbols which contribute 12 bars to the code.

The error correcting code specified for each of the bar codes is a Reed-Solomon code over the field GF(64) with 64 elements. We use a primitive element α of the field GF(64) which is a root of X^6+X+1 (over GF(2)). This means that the 63 powers α^i, for i=0, 1, . . . , 62 are the 63 distinct non-zero elements of the field. Also the 6 elements $\alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5$ form a linear basis for the field GF(64). Thus each element w of GF(64) has a unique expression as a sum,

$$(*)\quad w=w_5\alpha^5+w_4\alpha^4+w_3\alpha^3+w_2\alpha^2+w_1\alpha^1+w_0\alpha^0$$

where each w_i=0 or 1. By using the identity $\alpha^6=\alpha+1$ any power of α can be expressed in the form (*).

The translation from bar to field elements is accomplished by grouping the bars in sets of three. Each bar corresponds to a pattern of 2 bits:

H=00, A=01, D=10, T=11

A set of three bars corresponds to 6 bits which we take as the values of w_i in the expression (*). So for example,

$$TDT \rightarrow 11\ 10\ 11 \rightarrow 1\alpha^5+1\alpha^4+1\alpha^3+0\alpha^2+1\alpha^1+1\alpha^0=\alpha^{21}$$

Table 3 displays the correspondence between bar patterns and field elements. In the encoding and decoding software the field elements are represented as integers in the range 0 to 63, (thus as 6 bits). For example, the element α^{21}, above, which has bit pattern 111011 corresponds to the integer 59 which is 111011 in binary (59=32+16+8+2+1). In Table 3, the column int gives the integer corresponding to each field element in this way.

TABLE 3

bars	i	α^1	int	bars	i	α^1	int
HHH	**	000000	0	DAA	31	100101	37
HHA	0	000001	1	HDA	32	001001	9
HHD	1	000010	2	AHD	33	010010	18
HAH	2	000100	4	DAH	34	100100	16
HDH	3	001000	8	HDT	35	001011	11
AHH	4	010000	16	AAD	36	010110	22
DHH	5	100000	32	DTH	37	101100	44
HHT	6	000011	3	ADT	38	011011	27
HAD	7	000110	6	TAD	39	110110	54
HTH	8	001100	12	DTT	40	101111	47
ADH	9	011000	24	ATA	41	011101	29
THH	10	110000	48	TDD	42	111000	58
DHT	11	100011	35	TAT	43	110111	55
HAA	12	000101	5	DTA	44	101101	45
HDD	13	001010	10	ADA	45	011001	25
AAH	14	010100	20	THD	46	110010	50
DDH	15	101000	40	DAT	47	100111	39
AHT	16	010011	19	HTA	48	001101	13
DAD	17	100110	38	ADD	49	011010	26
HTT	18	001111	15	TAH	50	110100	52
ATD	19	011110	30	DDT	51	101011	43
TTH	20	111100	60	AAA	52	010101	21
TDT	21	111011	59	DDD	53	101010	42
TAA	22	110101	53	AAT	54	010111	23
DDA	23	101001	41	DTD	55	101110	46
AHA	24	010001	17	ATT	56	011111	31
DHD	25	100010	34	TTD	57	111110	62
HAT	26	000111	7	TTT	58	111111	63
HTD	27	001110	14	TTA	59	111101	61
ATH	28	011100	28	TDA	60	111001	57
TDH	29	111000	56	THA	61	110001	49
THT	30	110011	51	DHA	62	100001	33

The coded portion is divided into blocks of three bars. Each block represents one element of the field. The ECC runs from right to left across the bars. We set $c_i = b_{47-3i}b_{48-3i}b_{49-3i}$.

Thus we have:

$$\underbrace{b_0b_1}_{\text{start}}\ \underbrace{b_2b_3b_4}_{c_{15}}\ \underbrace{b_5b_6b_7}_{c_{14}} \ldots \underbrace{b_{44}b_{45}b_{46}}_{c_1}\ \underbrace{b_{47}b_{48}b_{49}}_{c_0}\ \underbrace{b_{50}b_{51}b_{52}b_{53}}_{\text{machine ID}}\ \underbrace{b_{54}b_{55}}_{\text{stop}}$$

Using these 16 elements $c_0, \ldots, c_{15}$ of GF(64) the ECC is defined by specifying that a codeword must satisfy:

$$(**) \qquad \sum_{i=0}^{15} c_i\alpha^{ij} = 0 \text{ for } j = 1, 2, \ldots, 10$$

The equation represents 10 equations with 10 unknown coefficients which is solved by the computer bar code generating software.

Using the symbology defined above, we record the postal code (PC) in bars $b_5 \ldots b_{19}$ hence in elements $c_{14}, \ldots, c_{10}$. The DCI goes in bars $b_2b_3b_4$ hence in c_{15}. The check symbols $c_0, \ldots, c_9$ are now uniquely determined by the parity checks (**).

For example, DCI=Z, postal code K1S 5B6, mach ID—DHAH encodes to the following codeword:

```
18  29  12  5  14  27  52  54  4  6  8  33  9  6  20  41
c0  ...                                          ...  c15
```

Table 2 then produces the bar code

AT DDA AAH HAD HDA DHA HDH HAD HAH TAD

TAH ADT HTD HAA HTH ATA AHD DHAH AT

Reorganized into A- and N- fields this is

$$\underbrace{\text{AT}}_{\text{Start}}\ \underbrace{\text{DDA}}_{\text{DCI}}\ \underbrace{\text{AAH HA DHD AD HAH DH}}_{\text{Postal Code -ANANAN}}$$

$$\underbrace{\text{HAD HAH TAD TAH ADT HTD HAA HTH ATA AHD}}_{\text{checks}}\ \underbrace{\text{DHAH}}_{\text{mach ID}}\ \underbrace{\text{AT}}_{\text{stop}}$$

For the CPC internal bar code, PostBar.C10, the 56 bars are arranged as follows:

$$\underbrace{b_0b_1}_{\text{start}}\ \underbrace{b_2b_3 \ldots b_{49}}_{\text{coded portion}}\ \underbrace{b_{50}b_{51}b_{52}b_{53}}_{\text{machine ID}}\ \underbrace{b_{54}b_{55}}_{\text{stop}}$$

The Domestic Bar Cods, PostBar.D22 is similar in definition to the CPC Internal code. All of the 79 bars of the Customer Code, except for the 4 start/stop bars are covered by the ECC. Again the bars correspond to field elements in sets of 3. Thus field element $c_i = b_{74-3i}b_{75-3i}b_{76-3i}$. We have:

$$\underbrace{b_0b_1}_{\text{start}}\ \underbrace{b_2b_3b_4}_{c_{24}}\ \underbrace{b_5b_6b_7}_{c_{23}}\ldots\underbrace{b_{71}b_{72}b_{73}}_{c_1}\ \underbrace{b_{74}b_{75}b_{76}}_{c_0}\ \underbrace{b_{77}b_{78}}_{\text{stop}}$$

Using these 25 elements $c_0, \ldots, c_{24}$ of GF(64) the ECC for the Customer Code is defined by specifying that a codeword must satisfy:

$$(***)\qquad \sum_{i=0}^{24} c_i\alpha^{ij} = 0 \text{ for } j = 1, 2, 3, 4$$

This equation represents 4 equations with 4 unknown coefficients which is solved by the computer bar code generating software.

Using the symbology defined above, we record the DCI in bars $b_2b_3b_4$ hence in the element c_{24}. The postal code is placed in bars b_5 to b_{19} hence in the elements $c_{23}, \ldots c_{19}$. The Address Locator (AL) goes in bars $b_{20}, \ldots b_{31}$ hence in $c_{18}, \ldots, c_{15}$. The customer field goes in bars $b_{32}, \ldots, b_{64}$ hence in $c_{14}, \ldots, c_4$. The check symbols $c_0, \ldots c_3$ are now uniquely determined by the parity checks (***).

For example, DCI=C, Postal Code=M4J 3W8, AL-1420, Cust field=ABCDEFGHIJK encodes the following codeword:

```
40  30  23  4  17  16  10  9  8  6  5  4  2  1  42  49  52  48  10  10  9  21  02
c0  ...                                                               ... c24
```

Using Table 3 to change from integer representation to bars produces the bar code:

```
AT HHD HHH AAA HDA HDD HDD THH TAH THA DDD HHA HHH
 HHD HAH HAA HAD HDH HDA HDD AHH AHA HAH AAT ATD DDH AT
```

This breaks down into A-, N-, and Z-fields as:

$$\underbrace{\text{AT}}_{\text{Start}}\ \underbrace{\text{HHD}}_{\text{DCI}}\ \underbrace{\text{HHH AA AHD AH DDH DD}}_{\text{Postal Code-ANANAN}}\ \underbrace{\text{THH TAH THA DDD}}_{\text{Address Locator}}$$

$$\underbrace{\text{HHA HHH HHD HAH HAA HAD HDH HDA HDD AHH}}_{\text{Customer Field}}\ \underbrace{\text{AHA HAH AAT ATD DDH}}_{\text{checks}}\ \underbrace{\text{AT}}_{\text{stop}}$$

The checks do not follow any of the Table 1 or 2 symbologies but are simply field elements coded as in Table 3.

It is noted that in a (16,6) Reed-Solomon code the probability of a random pattern being a valid codeword is $p=3.78\times10^{-6}$. Moreover, for PostBar.C10 the letters D,F,I, O,Q, or U currently do not appear in the bar codes and W or Z do not appear as the first letters of the code. There are 64 possible 3 bar system patterns, and only 20 are used for postal code letters. The digits for the postal code are encoded as 2 bar patterns, and only 10 of 16 are used. This gives a probability of:

$$\frac{18}{64}\times\frac{10}{16}\times\frac{20}{64}\times\frac{10}{16}\times\frac{20}{64}\times\frac{10}{16}=.0067$$

or about ½ of 1% that a random sequence of bars is a valid postal code. Combining this with p above gives a probability of 2.5×10^{-8} or 1 chance in 40,000,000 that a random string of bars will be interpreted as a valid postal code. Other internal checks, such as the use of the code type, will reduce this probability even further. This information should be incorporated into the OCR software.

FIGS. 7 through 11 respectively illustrate the format of PostBar.D07, D12, S06, S11 and G22. These structures will not be described herein as they will be readily understood from the foregoing description.

It should be understood that the inherent flexibility of the new code permits of many more applications than described and illustrated herein and the particular applications described are to be considered representative only.

Furthermore, it is envisaged that data compression techniques may be used to accommodate longer messages than currently provided for with PostBar.D07, D12 and D22. Data compression could also be used with the G and S codes.

Where data compression is used, it is likely the DCI will not be compressed so as to allow speedy determination of the make-up of the bar code. An example is the case of a service bar code present on a mail item that is being processed for sortation. As there is no sortation data in S codes, once the DCI is derived, the machine logic knows that no further decoding is required to process and decompress the data.

Suitable software listing for encoding of PostBar.C10, for encoding of PostBar.D07, D12 and D22, for decoding PostBar.C10, for decoding PostBar.D07, D12 and D22 follow:

```
Program Listing - 94-10-12
Encoder for Postbar.C10


#include <stdio.h>
#include <math.h>

/*Constants */
const int n = 16, m = 10; /*code length and number of check symbols */
const int N = 56; /* code length on bars */
const int bad_char = 35;/* code for bad output characters (print as "#") */
const int DCI_C10 = 90; /* DCI = "Z" for POSTBAR.C10 */

/* Define constants for printing bar codes (in printer units) */
const int xsize = 5;  /* width of the bars */
const int hsize = 60; /* height of Big bars */
const int asize = 22; /* height of ascender */
const int tsize = 15; /* track size */
const int dsize = 22; /* height of the descender */
const int xstep = 13; /* advance to next bar position */

/* Define constants for label printing (in printer units) */
const int nrows = 7;
const int ncols = 2;
const int xinit = 60;   /* left offset to first label */
const int yinit = 50;   /* down offset to first label */
const int xincr = 1240; /* horizontal label spacing */
const int yincr = 450;  /* vertical spacing of labels */
const int zincr = 8;    /* space between bars and text */

/*Global Variables */

/*Static Variables: defined by setup(); used by decode_the_bars() */
int prod[64] [64], inv[64], alog[64]; /*mult., inv. and anti-log. tables */
int Atable[26];  /* the 6 bit (field representation) of letters using A table */
int Aletter[64]; /* the inverse of the table Atable[] */
int Ntable[10];  /* the 4 bit representations of the digits using N table */
int Ndigit[16];  /* the inverse of Ntable[] */
int Ztable[37]; /* The 6 bit representations of alphanumerics using Z table */
int Zchar[64]; /* inverse of the Ztable[] */
int alpha[10]; /* defining roots for the code */

/*Dynamic Variables: output from encode_to_bars() */
int bars[56]; /* holds the bar code pattern in numbers */
char pcode[20];  /* holds the input string */

char postal_code[6];  /* holds the postal code in ascii */
char DCI; /* holds the DCI in ascii */
char AL[4]; /* holds the Address Locator in ascii */
char CustField[11]; /* holds the Customer Field in ascii */

/* Dynamic: additional inputs for printing */
char item_number[4];
char customer_name[25];
char inward_address[35];
char outward_address[30];

/* Function Prototypes */
void setup(void); /* builds the  tables;  */
void encode_to_bars(void); /* decodes the bars into postal data */
void new_pcode(char *); /* gets new data */
void initprn(void); /* initializes the printer */
void print_bars(int x, int y); /* generates bar codes at the given location */

int by_alpha(int u); /* utility that multiplies u by alpha in the finite field *
int shift(int u); /* utility that shifts ascii values for use in Atable, Ntable
int zshift(int u); /* utility that shifts ascii values for use in Ztable */

int verbose = 0; /* print debug information */
int output_bars = 0;

/* Main Program */

int main(argc,argv)

int argc;
char *argv[];

{
int i,count,curx,cury,num,nline;
int hh,vv;
int xx,yy,zz;
FILE *fd;
char *cp;
char buf[1024];

if (argc < 2) {
   printf("Encoder program for the C10 code. (batch version).\n");
   printf("This program prints 7x2 up labels on an HP Laserjet printer.\n\n");

   printf("Usage: bpenccl0 [-b] [-t] [-h #] [-v #] [-x #] [-y #] [-z #] filename
   printf("  h is the initial horizontal offset (default %d).\n",xinit);
   printf("  v is the initial vertical offset (default %d).\n",yinit);
   printf("  x is the inter-label horizontal spacing (default %d).\n",xincr);
   printf("  y is the inter-label vertical spacing (default %d).\n",yincr);
   printf("  z is the space between the bars and the text (default %d).\n",zincr
   printf("  # is dot positions in printer units (300 DPI).\n");
   printf("  t requests that information be printed to the screen.\n");
   printf("  b requests only bar code numbers be printed to the screen.\n");
   exit();
}
hh = xinit;
vv = yinit;
xx = xincr;
yy = yincr;


/* process command line arguments */

for (i=1; i<argc; i++)
   if (strcmp(argv[i],"-t") == 0)
      ++verbose;
   else if (strcmp(argv[i],"-b") == 0)
```

```
      ++output_bars;
      else if (strcmp(argv[i],"-h") == 0)
         hh = atoi(argv[++i]);
      else if (strcmp(argv[i],"-v") == 0)
         vv = atoi(argv[++i]);
      else if (strcmp(argv[i],"-x") == 0)
         xx = atoi(argv[++i]);
      else if (strcmp(argv[i],"-y") == 0)
         yy = atoi(argv[++i]);
      else if (strcmp(argv[i],"-z") == 0)
         zz = atoi(argv[++i]);
      else
         cp = argv[i];

   fd = fopen(cp,"r");
   if (!fd) {
      printf("Unable to open file %s.\n",cp);
      exit();
   }
   setup();

   if (verbose && !output_bars) {
      printf("=================================================\n");
      printf("August 15, 1994\n");
      printf("Encoding Program for the C10 code.\n");
   }

   if (!verbose && !output_bars) initprn();

   num = nrows * ncols; /* number of labels on the page */
   count = 0;
   nline = 0;
   while (fgets(buf,sizeof(buf),fd)) {
      ++nline;
      if (strlen(buf) < 116) {
         for (i=0; i<strlen(buf); ++i)
            if (buf[i] != ' ') break;
         if ((++i) == strlen(buf)) continue; /* skip blank lines */
         fprintf(stderr,"ERROR - Invalid input at line %d.\n",nline);
         exit();
      }
      new_pcode(buf);
      encode_to_bars();
      ++count;
      curx = hh + xx * (1 - (count % 2));
      cury = vv + yy * (int) (((count - 1) % num) / 2);
      if (verbose && !output_bars)
         printf("\nPrinting next label at (%d,%d).\n",curx,cury);
      print_bars(curx,cury);
         if ((count % num) == 0)
         if (!verbose && !output_bars) fprintf(stdprn,"%c",12);
   }
   if ((count % num))
      if (!verbose && !output_bars) fprintf(stdprn,"%c",12);
   }

/* ============================================================ */
/* Initialization Routines                                      */
/* ============================================================ */
```

```
void initprn(void)
{
int temp,i;

fprintf(stdprn,"%cE",27);            /* resets printer */
fprintf(stdprn,"%c*t300R",27);       /* seting up resolution */
fprintf(stdprn,"%c&lefloP",27);      /* sets various defaults */
fprintf(stdprn,"%c&alM",27);         /* more of the same */

}

int by_alpha(int u)
{
u = (u << 1);
if (u > 63) u =u^67;
return(u);
}

int shift(int u) /* changes ascii to indices for A and N tables */
{ if(u>64) u-=65;
else u -= 48;
return(u);
}

int zshift(int u) /* changes ascii to indices for Z table */
{ if(u>64) u-=65;
else if (u==32) u=36;
else u -= 22;
return(u);
}
void setup(void) /* The look-up tables are created */
{
int w0, w1;
int i, j, k, u, v;

/* Initialize the look-up symbology tables  */
/* The tables Atable[], Ntable[], Ztable[] convert a shifted ascii */
/* argument to the representations for the A,N and Z fields respectively. */
/* Shfted ascii means (ascii -65) for A .. Z and (ascii - 48) for 0..9 */

Atable[0] = 2; Atable[1] = 4; Atable[2] = 5; Atable[3] = 6;
Atable[4] = 8; Atable[5] = 9; Atable[6] = 10; Atable[7] = 1;
Atable[8] = 17; Atable[9] = 18; Atable[10] = 20; Atable[11] = 21;
Atable[12] = 0; Atable[13] = 24; Atable[14] = 25; Atable[15] = 26;
Atable[16] = 32; Atable[17] = 33; Atable[18] = 34; Atable[19] = 36;
Atable[20] = 37; Atable[21] = 38; Atable[22] = 40; Atable[23] = 41;
Atable[24] = 42; Atable[25] = 16;

for(k=0; k<64; k++) Aletter[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Aletter[Atable[k]] = k+65;

Ntable[0] = 0; Ntable[1] = 1; Ntable[2] = 2; Ntable[3] = 4; Ntable[4] = 5;
Ntable[5] = 6; Ntable[6] = 8; Ntable[7] = 9; Ntable[8] = 10; Ntable[9] = 12;

for(k=0; k<16; k++) Ndigit[k] = bad_char; /* unsuitable characters print as # */
for(k=0; k<10;k++) Ndigit[Ntable[k]] = k+48;

Ztable[0] = 0; Ztable[1] = 1; Ztable[2] = 2; Ztable[3] = 4; Ztable[4] = 5;
Ztable[5] = 6; Ztable[6] = 8; Ztable[7] = 9; Ztable[8] = 10; Ztable[9] = 16;
Ztable[10] = 17;Ztable[11] = 18;Ztable[12] = 20;Ztable[13] = 21;Ztable[14] = 22;
```

```
Ztable[15] = 24;Ztable[16] = 25;Ztable[17] = 26;Ztable[18] = 32;Ztable[19] = 33;
Ztable[20] = 34;Ztable[21] = 36;Ztable[22] = 37;Ztable[23] = 38;Ztable[24] = 40;
Ztable[25] = 41;Ztable[26] = 42;Ztable[27] = 48;Ztable[28] = 49;Ztable[29] = 50;
Ztable[30] = 52;Ztable[31] = 53;Ztable[32] = 54;Ztable[33] = 56;Ztable[34] = 57;
Ztable[35] = 58;Ztable[36] = 3;

for(k=0; k < 64; k++) Zchar[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Zchar[Ztable[k]] = k + 65;  /* ascii for the letters */
for(k=26; k<36; k++) Zchar[Ztable[k]] = k + 22; /* ascii for the digits */
Zchar[3] = 32; /* ascii for a space */

/* the prod[][] and inv[] tables are defined */
for(k = 0; k < 64;k++) {prod[0][k] = 0; prod[1][k] = k;}

w0 = 1; alog[0] = 1;
for(u=1;u<64;u++)
        {w1 = by_alpha(w0);
        alog[u] = w1;
        for(v=0;v<64;v++)
                {
                prod[w1][v] = by_alpha(prod[w0][v]);
                if (prod[w1][v] == 1) inv[w1] = v;
                }
        w0 = w1;}

/* These are the codes for the roots of the generator poly. */
/* They are alpha^1, ... , alpha^10. */
for (k=0;k<m;k++) alpha[k] = alog[k+1];

return;
}

/* ============================================================ */
/* Makes Input for the Encoder */
/* ============================================================ */

void new_pcode(char *buf)
{
strncpy (item_number,buf,4);
strncpy (customer_name,&buf[4],25);
strncpy (inward_address,&buf[29],35);
strncpy (outward_address,&buf[64],30);
strncpy (postal_code,&buf[94],6);
DCI = buf[100];
strncpy (AL,&buf[101],4);
strncpy (CustField,&buf[105],11);

/* Transfer the data to pcode for further processing */

pcode[0] = DCI_C10;
strncpy (&pcode[1],postal_code,6);
strncpy (&pcode[7],AL,4);
return;
}

void encode_to_bars(void)
{/* This routine encodes pcode[] as a bar pattern in bars[] */
int i,j,k,zeta; /*loop variables */
int deg_gen; /* holds the degree of the generator polynomial */
int mess[21]; /*the 21 message or information symbols */
int gen[24]; /*the generator polynomial */
int codeword[26]; /* codeword produced by the encoder */
int bars_copy[79];/* holds a copy of the bar pattern */

/*============================================================ */
/*                Data Converter */
/* The DCI, postal code and data fields (if any) */
/* are converted to a series of n-m field elements */
/* placed in mess[0] .. mess[n-m-1] ready for the encoder. */

/* DCI is put in mess[n-m-1] */
mess[n-m-1] = Ztable[zshift((int) pcode[0])];

/* Repack the Postal Code */
mess[n-m-2] = Atable[shift((int) pcode[1])];
mess[n-m-3] = (Ntable[shift((int) pcode[2])]<<2)^(Atable[shift((int) pcode[3])]>
mess[n-m-4] = ((Atable[shift((int) pcode[3])]&15)<<2)^(Ntable[shift((int) pcode[
mess[n-m-5] = ((Ntable[shift((int) pcode[4])]&3)<<4)^(Atable[shift((int) pcode[5
mess[n-m-6] = ((Atable[shift((int) pcode[5])]&3)<<4)^Ntable[shift((int) pcode[6]

for(j=n-m-7;j>=0;j--) mess[j] = Ztable[zshift((int)pcode[n-m-j])];


/*====================================================================== */
/*                       Error Correction Encoder */
/* The message is in the form of n-m field elements mess[0] ... mess[n-m-1]. */
/* These go into codeword[m]...codeword[n-1], while the check */
/* symbols go into codeword[0]...codeword[m-1]. */

for(i=0;i<m;i++) codeword[i] = 0;
for(i=m;i<n;i++) codeword[i] = mess[i-m];

/* The generator polynomial is defined. */
deg_gen = 0;gen[0]=1;
for (j = 0;j<m;j++)
        {deg_gen++; gen[deg_gen] = 1;
        for(i=deg_gen-1;i >0;i--) gen[i] = prod[gen[i]][alpha[j]]^gen[i-1];
        gen[0] = prod[gen[0]][alpha[j]];
        }

for(zeta = n-m-1; zeta >=0; zeta--)
        {
        for(k = 0; k <= m ; k++)
                {codeword[zeta + k]
                         = codeword[zeta + k] ^ (prod[gen[k]][codeword[m + zeta]]
                }
        }
for(i=m;i<n;i++) codeword[i] = mess[i-m]; /* restore the high order symbols */

/*============================================================ */
/*                           Barcode Maker */
/*============================================================ */
/* This routine takes codeword[0..n-1] and produces the barcode pattern. */
/* The pattern is placed in bars[] as numbers from {0,1,2,3}={H,A,D,T} */

/* The start and stop bars are put in place. */
bars[0] = bars[N-2] = 1;
bars[1] = bars[N-1] = 3;

/* Put the machine ID bars in bars[50],...,bars[53] coded as 0,1,2,3 */
for(i=0; i < 4; i++)
        switch((int)pcode[i+7])
                 {case 72: {bars[50+i] = 0;break;}
                  case 68: {bars[50+i] = 2;break;}
                  case 84: {bars[50+i] = 3;break;}
                  case 65: {bars[50+i] = 1;}
                 }
/* The codeword[] is made into bars starting with codeword[n-1] */
/* and working down. */
/* This puts the checks at the right hand end of the barcode. */

for(j=2,k=n-1; k>=0;j+=3,k--)
        {bars[j] = codeword[k] >> 4;
        bars[j+1] = (codeword[k]>>2)&3;
        bars[j+2] = codeword[k]&3;
        }
return;
}

/* ================================================== */
/* Print the Barcode  */
/* ================================================== */
/* This routine prints out the bar pattern using vertical rules */

void print_bars(int x, int y)
{
int j,ht,yy;

if (verbose || output_bars) {
  for (j = 0; j < N; j++) printf("%d",bars[j]);printf("\n");
  return;
}

for (j=0; j<N; j++) {
        if (bars[j] == 0) {
                 ht = hsize;
                 yy = y - asize - tsize;
        } else if (bars[j] == 1) {
                 ht = asize + tsize;
                 yy = y - asize - tsize;
        } else if (bars[j] == 2) {
                 ht = tsize + dsize;
                 yy = y - tsize;
        } else if (bars[j] == 3) {
                 ht = tsize;
                 yy = y - tsize;
        }
        fprintf(stdprn,"%c*p%dx%dY",27,x,yy);
        fprintf(stdprn,"%c*c%da%db0P",27,xsize,ht);
        x = x + xstep;
}
}
```

```
Program Listing - 94-10-12
Encoder for Postbar.D07, Postbar.D12 and Postbar.D22.


#include <stdio.h>
#include <math.h>

/* Constants */
const int m = 4; /* number of check symbols */
const int bad_char = 35; /* code for bad characters (print as "#") */

#define DCI_D07 65
#define DCI_D12 66
#define DCI_D22 67

/* Define constants for printing bar codes (in printer units) */
const int xsize = 5;  /* width of the bars */
const int hsize = 60; /* height of Big bars */
const int asize = 22; /* height of ascender */
const int tsize = 15; /* track size */
const int dsize = 22; /* height of the descender */
const int xstep = 13; /* advance to next bar position */

/* Define constants for label printing (in printer units) */
const int nrows = 7;
const int ncols = 2;
const int xinit = 60;   /* left offset to first label */
const int yinit = 50;   /* down offset to first label */
const int xincr = 1240; /* horizontal label spacing */
const int yincr = 450;  /* vertical spacing of labels */
const int zincr = 8;    /* space between bars and text */

/* Global Variables */

/* Static Variables: defined by setup(); used by decode_the_bars() */
int prod[64] [64], inv[64], alog[64]; /* mult., inv. and anti-log. tables */
int Atable[26];  /* the 6 bit (field representation) of letters using A table */
int Aletter[64]; /* the inverse of the table Atable[] */
int Ntable[10];  /* the 4 bit representations of the digits using N table */
int Ndigit[16];  /* the inverse of Ntable[] */
int Ztable[37]; /* The 6 bit representations of alphanumerics using Z table */
int Zchar[64]; /* inverse of the Ztable[] */
int alpha[4]; /* defining roots for the error correcting code */

/* Dynamic Variables: input/output for encode_to_bars() */
int bars[79]; /* holds the bar code pattern in numbers */
char pcode[40];  /* holds the input string */

char postal_code[6];  /* holds the postal code in ascii */
char DCI; /* holds the DCI in ascii */
char AL[4]; /* holds the Address Locator in ascii */
char CustField[11]; /* holds the Customer Field in ascii */

                  :
```

```
/* Dynamic additional inputs for printing */
char item_number[4];
char customer_name[25];
char inward_address[35];
char outward_address[30];

/* Function Prototypes */

/* principal routines */
void setup(void); /* builds the tables; */
void encode_to_bars(void); /* encodes the data as a barcode D07, D12 or D22 */
void new_pcode(char *); /* gets new data */
void initprn(void); /* initializes the printer */
void print_bars(int x, int y); /* generates bar codes at the given location */
void print_text(int x, int y, int z); /* prints name and address information */
/* utilities used by setup() */
int by_alpha(int u); /* utility that multiplies u by alpha in the finite field *
int shift(int u); /* utility that shifts ascii values for use in Atable, Ntable
int zshift(int u); /* utility that shifts ascii values for use in Ztable */

int verbose = 0; /* print debug information */
int output_bars = 0;

/* Main Program */

int main(argc,argv)

int argc;
char *argv[];

{
int i,count,curx,cury,num,nline;
int hh,vv;
int xx,yy,zz;
FILE *fd;
char *cp;
char buf[1024];

if (argc < 2) {
   printf("Encoder program for the D code family (batch version).\n");
   printf("This program prints 7x2 up labels on an HP Laserjet printer.\n\n");
   printf("Usage: bpencd [-b] [-t] [-h #] [-v #] [-x #] [-y #] [-z #] filename\n
   printf("  h is the initial horizontal offset (default %d).\n",xinit);
   printf("  v is the initial vertical offset (default %d).\n",yinit);
   printf("  x is the inter-label horizontal spacing (default %d).\n",xincr);
   printf("  y is the inter-label vertical spacing (default %d).\n",yincr);
   printf("  z is the space between the bars and the text (default %d).\n",zincr
   printf("  # is dot positions in printer units (300 DPI).\n");
   printf("  t requests that information be printed to the screen.\n");
   printf("  b requests only bar code numbers be printed to the screen.\n");
   exit();
}
hh = xinit;
vv = yinit;
xx = xincr;
yy = yincr;
zz = zincr;

/* process command line arguments */
```

```
  for (i=1; i<argc; i++)
     if (strcmp(argv[i],"-t") == 0)
        ++verbose;
     else if (strcmp(argv[i],"-b") == 0)
        ++output_bars;
     else if (strcmp(argv[i],"-h") == 0)
        hh = atoi(argv[++i]);
     else if (strcmp(argv[i],"-v") == 0)
        vv = atoi(argv[++i]);
     else if (strcmp(argv[i],"-x") == 0)
        xx = atoi(argv[++i]);
     else if (strcmp(argv[i],"-y") == 0)
        yy = atoi(argv[++i]);
     else if (strcmp(argv[i],"-z") == 0)
        zz = atoi(argv[++i]);
     else
        cp = argv[i];

  fd = fopen(cp,"r");
  if (!fd) {
     printf("Unable to open file %s.\n",cp);
     exit();
  }
  setup();

  if (verbose && !output_bars) {
     printf("==================================================\n");
     printf("August 15, 1994\n");
     printf("Encoding Program for the D codes.\n");
  }

  if (!verbose && !output_bars) initprn();

  num = nrows * ncols; /* number of labels on the page */
  count = 0;
  nline = 0;
  while (fgets(buf,sizeof(buf),fd)) {
     ++nline;
     if (strlen(buf) < 116) {
        for (i=0; i<strlen(buf); ++i)
           if (buf[i] != ' ') break;
        if ((++i) == strlen(buf)) continue; /* skip blank lines */
        fprintf(stderr,"ERROR - Invalid input at line %d.\n",nline);
        exit();
     }
     new_pcode(buf);
     encode_to_bars();
     ++count;
     curx = hh + xx * (1 - (count % 2));
     cury = vv + yy * (int) (((count - 1) % num) / 2);
     if (verbose && !output_bars)
        printf("\nPrinting next label at (%d,%d).\n",curx,cury);
     print_bars(curx,cury);
     if (!output_bars) print_text(curx,cury,zz);
     if ((count % num) == 0)
        if (!verbose && !output_bars) fprintf(stdprn,"%c",12);
  }
  if ((count % num))
     if (!verbose && !output_bars) fprintf(stdprn,"%c",12);
  }
```

```
/* ================================================================= */
/* Initialization Routines                                            */
/* ================================================================= */

void initprn(void)
{
int temp,i;

fprintf(stdprn,"%cE",27);            /* resets printer */
fprintf(stdprn,"%c*t300R",27);       /* seting up resolution */
fprintf(stdprn,"%c&lefloP",27);      /* sets various defaults */
fprintf(stdprn,"%c&alM",27);         /* more of the same */

}

int by_alpha(int u)
{
u = (u << 1);
if (u > 63) u =u^67;
return(u);
}

int shift(int u) /* changes ascii to indices for A and N tables */
{ if(u>64) u-=65;
else u -= 48;
return(u);
}

int zshift(int u) /* changes ascii to indices for Z table */
{ if(u>64) u-=65;
else if (u==32) u=36;
else u -= 22;
return(u);
}

void setup(void) /* The look-up tables are created */
{
int w0, w1;
int i, j, k, u, v;

/* Initialize the look-up symbology tables  */
/* The tables Atable[], Ntable[], Ztable[] convert a shifted ascii */
/* argument to the representations for the A,N and Z fields respectively. */
/* Shfted ascii means (ascii -65) for A .. Z and (ascii - 48) for 0..9 */

Atable[0] = 2; Atable[1] = 4; Atable[2] = 5; Atable[3] = 6;
Atable[4] = 8; Atable[5] = 9; Atable[6] = 10; Atable[7] = 1;
Atable[8] = 17; Atable[9] = 18; Atable[10] = 20; Atable[11] = 21;
Atable[12] = 0; Atable[13] = 24; Atable[14] = 25; Atable[15] = 26;
Atable[16] = 32; Atable[17] = 33; Atable[18] = 34; Atable[19] = 36;
Atable[20] = 37; Atable[21] = 38; Atable[22] = 40; Atable[23] = 41;
Atable[24] = 42; Atable[25] = 16;

for(k=0; k<64; k++) Aletter[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Aletter[Atable[k]] = k+65;

Ntable[0] = 0; Ntable[1] = 1; Ntable[2] = 2; Ntable[3] = 4; Ntable[4] = 5;
Ntable[5] = 6; Ntable[6] = 8; Ntable[7] = 9; Ntable[8] = 10; Ntable[9] = 12;
```

```
for(k=0; k<16; k++) Ndigit[k] = bad_char; /* unsuitable characters print as # */
for(k=0; k<10;k++) Ndigit[Ntable[k]] = k+48;

Ztable[0] = 0; Ztable[1] = 1; Ztable[2] = 2; Ztable[3] = 4; Ztable[4] = 5;
Ztable[5] = 6; Ztable[6] = 8; Ztable[7] = 9; Ztable[8] = 10; Ztable[9] = 16;
Ztable[10] = 17;Ztable[11] = 18;Ztable[12] = 20;Ztable[13] = 21;Ztable[14] = 22;
Ztable[15] = 24;Ztable[16] = 25;Ztable[17] = 26;Ztable[18] = 32;Ztable[19] = 33;
Ztable[20] = 34;Ztable[21] = 36;Ztable[22] = 37;Ztable[23] = 38;Ztable[24] = 40;
Ztable[25] = 41;Ztable[26] = 42;Ztable[27] = 48;Ztable[28] = 49;Ztable[29] = 50;
Ztable[30] = 52;Ztable[31] = 53;Ztable[32] = 54;Ztable[33] = 56;Ztable[34] = 57;
Ztable[35] = 58;Ztable[36] = 3;

for(k=0; k < 64; k++) Zchar[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Zchar[Ztable[k]] = k + 65;  /* ascii for the letters */
for(k=26; k<36; k++) Zchar[Ztable[k]] = k + 22; /* ascii for the digits */
Zchar[3] = 32; /* ascii for a space */

/* the prod[][] and inv[] tables are defined */
for(k = 0; k < 64;k++) {prod[0][k] = 0; prod[1][k] = k;}

w0 = 1; alog[0] = 1;
for(u=1;u<64;u++)
        {w1 = by_alpha(w0);
        alog[u] = w1;
        for(v=0;v<64;v++)
                {
                prod[w1][v] = by_alpha(prod[w0][v]);
                if (prod[w1][v] == 1) inv[w1] = v;
                }
        w0 = w1;}

/* These are the codes for the roots of the generator poly. */
/* They are alpha^1, ... , alpha^10. */
for (k=0;k<m;k++) alpha[k] = alog[k+1];

return;
}
/* ============================================== */
/* Makes Input for the Encoder */
/* ============================================== */

void new_pcode(char *buf)
{
int i;

strncpy (item_number,buf,4);
strncpy (customer_name,&buf[4],25);
strncpy (inward_address,&buf[29],35);
strncpy (outward_address,&buf[64],30);
strncpy (postal_code,&buf[94],6);
DCI = buf[100];
strncpy (AL,&buf[101],4);
strncpy (CustField,&buf[105],11);

/* Transfer the data to pcode for further processing */

pcode[0] = DCI; /* Assume input record contains valid DCI */
strncpy (&pcode[1],postal_code,6);
strncpy (&pcode[7],AL,4);
strncpy (&pcode[11],CustField,11);
```

```
  if (!verbose) return;

  switch(pcode[0])
          {case DCI_D07 : printf("\nUsing code POSTBAR.D07");break;
          case DCI_D12 : printf("\nUsing code POSTBAR.D12");break;
          case DCI_D22 : printf("\nUsing code POSTBAR.D22");
          }
 printf("  DCI = %c",(char)pcode[0]);
 return;
 }

 /* ============================================================ */
 /* The main Encoder Routine */
 /* ============================================================ */

 void encode_to_bars(void)
 {int i,j,k,zeta; /* loop variables */
 int n,N;/* code length in symbols and bars */
 int mess[21]; /* the 21 message or information symbols */
 int gen[26]; /* the generator polynomial */
 int deg_gen; /* holds the degree of the generator polynomial */
 int codeword[26]; /* codeword produced by the encoder */

 switch (pcode[0])
         {case DCI_D07: {n=10;break;}
          case DCI_D12: {n=15;break;}
          case DCI_D22:  n=25;}

 N = 3*n + 4; /* total number of bars in the code used */
 /* ============================================================ */
 /* Data Converter */
 /* The DCI, postal code and data fields (if any) */
 /* are converted to a series of n-m field elements */
 /* placed in mess[0] .. mess[n-m-1] ready for the encoder. */

 /* DCI is put in mess[n-m-1] */
 mess[n-m-1] = Ztable[zshift((int) pcode[0])];

 /* Repack the Postal Code */
 mess[n-m-2] = Atable[shift((int) pcode[1])];
 mess[n-m-3] = (Ntable[shift((int) pcode[2])]<<2)^(Atable[shift((int) pcode[3])]>
 mess[n-m-4] = ((Atable[shift((int) pcode[3])]&15)<<2)^(Ntable[shift((int) pcode[
 mess[n-m-5] = ((Ntable[shift((int) pcode[4])]&3)<<4)^(Atable[shift((int) pcode[5
 mess[n-m-6] = ((Atable[shift((int) pcode[5])]&3)<<4)^Ntable[shift((int) pcode[6]

 for(j=n-m-7;j>=0;j--) mess[j] = Ztable[zshift((int)pcode[n-m-j])];


 /* ============================================================ */
 /* Error Correction Encoder */
 /* The message is in the form of n-m field elements mess[0] ... mess[n-m-1]. */
 /* These go into codeword[m]...codeword[n-1], while the check */
 /* symbols go into codeword[0]...codeword[m-1]. */

 for(i=0;i<m;i++) codeword[i] = 0;
 for(i=m;i<n;i++) codeword[i] = mess[i-m];

/* The generator polynomial is defined. */
deg_gen = 0;gen[0]=1;
```

```
 for (j = 0;j<m;j++)
          {deg_gen++; gen[deg_gen] = 1;
          for(i=deg_gen-1;i >0;i--) gen[i] = prod[gen[i]][alpha[j]]^gen[i-1];
          gen[0] = prod[gen[0]][alpha[j]];
          }

 for(zeta = n-m-1; zeta >=0; zeta--)
          {
          for(k = 0; k <= m ; k++)
                   {
                   codeword[zeta + k] = codeword[zeta + k] ^ (prod[gen[k]][codeword
                   }
          }
 for(i=m;i<n;i++) codeword[i] = mess[i-m]; /* restore the high order symbols */

/* ================================================= */
/* Barcode Maker */
/* ================================================= */
/* This routine takes codeword[0..n-1] and produces the barcode pattern. */
/* The pattern is placed in bars[] as numbers from {0,1,2,3} */
/* The array display[] converts from 0,1,2,3 to B,U,D,T */
/* The start and stop bars are put in place. */
bars[0] = bars[N-2] = 1;
bars[1] = bars[N-1] = 3;

/* The codeword[] is made into bars starting with codeword[n-1] */
/* and working down. */
/* This puts the checks at the right hand end of the barcode. */

for(j=2,k=n-1; k>=0;j+=3,k--)
         {bars[j] = codeword[k] >> 4;
         bars[j+1] = (codeword[k]>>2)&3;
         bars[j+2] = codeword[k]&3;
         }
return;
}

/* ================================================= */
/* Print the Barcode  */
/* ================================================= */
/* This routine prints out the bar pattern using vertical rules */

void print_bars(int x, int y)
{
int j,ht,yy;
int N; /* code length in bars */

switch(pcode[0])
         {case DCI_D07 :{N=34;break;}
          case DCI_D12 : {N=49;break;}
          case DCI_D22 : {N=79;}}

if (verbose || output_bars) {
  for (j = 0; j < N; j++) printf("%d",bars[j]);printf("\n");
  return;
}

for (j=0; j<N; j++) {
        if (bars[j] == 0) {
                ht = hsize;
                yy = y - asize - tsize;
        } else if (bars[j] == 1) {
                ht = asize + tsize;
                yy = y - asize - tsize;
        } else if (bars[j] == 2) {
                ht = tsize + dsize;
                yy = y - tsize;
        } else if (bars[j] == 3) {
                ht = tsize;
                yy = y - tsize;
        }
        fprintf(stdprn,"%c*p%dx%dY",27,x,yy);
        fprintf(stdprn,"%c*c%da%db0P",27,xsize,ht);
        x = x + xstep;
 }
 }
/* ================================================ */
/* Print the text information                       */
/* ================================================ */
/* This routine prints the name and address information */

void print_text(int x, int y, int zz)
{
int i;
char buf[40];

y = y + dsize + zz + 30;

strncpy(buf,customer_name,25);
i = 24;
while (i && buf[i] == ' ') --i;
buf[++i] = '\0';

if (verbose) printf("%s\n",buf);
else {
   fprintf(stdprn,"%c*p%dx%dY",27,x,y);
   fprintf(stdprn,"%s",buf);
}

y = y + 60;

strncpy(buf,inward_address,35);
i = 34;
while (i && buf[i] == ' ') --i;
buf[++i] = '\0';

if (verbose) printf("%s\n",buf);
else {
   fprintf(stdprn,"%c*p%dx%dY",27,x,y);
   fprintf(stdprn,"%s",buf);
}

y = y + 60;

i = 29;
while (i && outward_address[i] == ' ') --i;

strncpy(buf,outward_address,i+1);
buf[++i] = ' ';
buf[++i] = ' ';
```

```
strncpy(&buf[++i],postal_code,3);
i = i + 3;
buf[i++] = ' ';
strncpy(&buf[i],&postal_code[3],3);
i = i + 3;
buf[i] = '\0';

if (verbose) printf("%s\n",buf);
else {
   fprintf(stdprn,"%c*p%dx%dY",27,x,y);
   fprintf(stdprn,"%s",buf);
}
}
```

Program Listing - 94-10-12
Decoder for Postbar.C10

```
#include <stdio.h>
#include <math.h>

/*Constants */
const int n = 16, m = 10; /*code length and number of check symbols */
const int N = 56; /* code length on bars */
const int erase_max = 10; /* max. number of erasure corrections allowed */
const int bad_char = 35;/* code for bad output characters (print as "#") */
/*Global Variables */

/*Static Variables: defined by setup(); used by decode_the_bars() */
int prod[64] [64], inv[64], alog[64]; /*mult., inv. and anti-log. tables */
int Atable[26];  /* the 6 bit (field representation) of letters using A table */
int Aletter[64]; /* the inverse of the table Atable[] */
int Ntable[10];  /* the 4 bit representations of the digits using N table */
int Ndigit[16];  /* the inverse of Ntable[] */
int Ztable[37]; /* The 6 bit representations of alphanumerics using Z table */
int Zchar[64]; /* inverse of the Ztable[] */
int alpha[10]; /* defining roots for the code */

/*Dynamic Variables: input to decode_the_bars() */
int bars[56]; /* holds the bar code pattern in numbers */
/*Dynamic Variables: output from decode_the_bars */
int DCI; /* holds the DCI in ascii after decoding */
int mach_id[4]; /* holds the Machine ID as 0,1,2,3 after decoding */
int decode_flag; /* flag indicating the result of the decoding */
int post_code[6]; /* ascii for the 6 chars. of the postal code after decoding */

/* Function Prototypes */
void setup(void); /* builds the  tables;  */
void get_decoder_input(char *); /* makes test input for decode_the_bars */
void decode_the_bars(void); /* decodes the bars into postal data */
void print_data(void); /* prints the output data */

int by_alpha(int u); /* utility that multiplies u by alpha in the finite field *
int shift(int u); /* utility that shifts ascii values for use in Atable, Ntable
int zshift(int u); /* utility that shifts ascii values for use in Ztable */

int main(argc,argv)

int argc;
char *argv[];

{
FILE *fd;
char *cp;
char buf[1024];
int i,nline;

if (argc < 2) {
      printf("Decoder program for code C10 (batch version).\n\n");
      printf("Usage: bpdecc10 filename\n\n");
      printf("  filename contains barcodes to be decoded.\n");
      exit();
   }

   cp = argv[1];

   fd = fopen(cp,"r");
   if (!fd) {
      printf("Unable to open file %s.\n",cp);
      exit();
   }

   setup();

   nline = 0;
   while (fgets(buf,sizeof(buf),fd)) {
      ++nline;
      if (buf[strlen(buf)-1] == '\n') buf[strlen(buf)-1] = '\0';
      for (i=0; i<strlen(buf); i++) {
         if (buf[i] < '0' || buf[i] > '4') {
            fprintf(stderr,"ERROR - Invalid input at line %d.\n",nline);
            fprintf(stderr,"        Invalid character at position %d.\n",i);
            exit();
         }
      }
      get_decoder_input(buf);
      decode_the_bars();
      print_data();
   }
   }

  /*============================================================== */
  /*                         Initialization Routines */
  /*============================================================== */


  int by_alpha(int u)
  {
  u = (u << 1);
  if (u > 63) u =u^67;
  return(u);
  }

  int shift(int u) /* changes ascii to indices for A and N tables */
  { if(u>64) u-=65;
  else u -= 48;
  return(u);
  }

  int zshift(int u) /* changes ascii to indices for Z table */
  { if(u>64) u-=65;
  else if (u==32) u=36;
  else u -= 22;
  return(u);
  }
  void setup(void) /* The look-up tables are created */
  {
  int w0, w1;
```

```
int i, j, k, u, v;

/* Initialize the look-up symbology tables  */
/* The tables Atable[], Ntable[], Ztable[] convert a shifted ascii */
/* argument to the representations for the A,N and Z fields respectively. */
/* Shfted ascii means (ascii -65) for A .. Z and (ascii - 48) for 0..9 */

Atable[0] = 2; Atable[1] = 4; Atable[2] = 5; Atable[3] = 6;
Atable[4] = 8; Atable[5] = 9; Atable[6] = 10; Atable[7] = 1;
Atable[8] = 17; Atable[9] = 18; Atable[10] = 20; Atable[11] = 21;
Atable[12] = 0; Atable[13] = 24; Atable[14] = 25; Atable[15] = 26;
Atable[16] = 32; Atable[17] = 33; Atable[18] = 34; Atable[19] = 36;
Atable[20] = 37; Atable[21] = 38; Atable[22] = 40; Atable[23] = 41;
Atable[24] = 42; Atable[25] = 16;

for(k=0; k<64; k++) Aletter[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Aletter[Atable[k]] = k+65;

Ntable[0] = 0; Ntable[1] = 1; Ntable[2] = 2; Ntable[3] = 4; Ntable[4] = 5;
Ntable[5] = 6; Ntable[6] = 8; Ntable[7] = 9; Ntable[8] = 10; Ntable[9] = 12;

for(k=0; k<16; k++) Ndigit[k] = bad_char; /* unsuitable characters print as # */
for(k=0; k<10;k++) Ndigit[Ntable[k]] = k+48;

Ztable[0] = 0; Ztable[1] = 1; Ztable[2] = 2; Ztable[3] = 4; Ztable[4] = 5;
Ztable[5] = 6; Ztable[6] = 8; Ztable[7] = 9; Ztable[8] = 10; Ztable[9] = 16;
Ztable[10] = 17;Ztable[11] = 18;Ztable[12] = 20;Ztable[13] = 21;Ztable[14] = 22;
Ztable[15] = 24;Ztable[16] = 25;Ztable[17] = 26;Ztable[18] = 32;Ztable[19] = 33;
Ztable[20] = 34;Ztable[21] = 36;Ztable[22] = 37;Ztable[23] = 38;Ztable[24] = 40;
Ztable[25] = 41;Ztable[26] = 42;Ztable[27] = 48;Ztable[28] = 49;Ztable[29] = 50;
Ztable[30] = 52;Ztable[31] = 53;Ztable[32] = 54;Ztable[33] = 56;Ztable[34] = 57;
Ztable[35] = 58;Ztable[36] = 3;

for(k=0; k < 64; k++) Zchar[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Zchar[Ztable[k]] = k + 65;  /* ascii for the letters */
for(k=26; k<36; k++) Zchar[Ztable[k]] = k + 22; /* ascii for the digits */
Zchar[3] = 32; /* ascii for a space */

/* the prod[][] and inv[] tables are defined */
for(k = 0; k < 64;k++) {prod[0][k] = 0; prod[1][k] = k;}

w0 = 1; alog[0] = 1;
for(u=1;u<64;u++)
        {w1 = by_alpha(w0);
        alog[u] = w1;
        for(v=0;v<64;v++)
                {
                prod[w1][v] = by_alpha(prod[w0][v]);
                if (prod[w1][v] == 1) inv[w1] = v;
                }
        w0 = w1;}

/* These are the codes for the roots of the generator poly. */
/* They are alpha^1, ... , alpha^10. */
for (k=0;k<m;k++) alpha[k] = alog[k+1];

return;
}
/*============================================== */
/*                 Gets Input for Decoder        */
```

```
/*=========================================== */
void get_decoder_input(char *buf)
{
int i;

for (i=0; i<strlen(buf); i++) bars[i] = buf[i] - '0';
return;
}

/*=================================================== */
/*                    Decode the Bars  */
/*=================================================== */
/* The bar pattern is received in bars[] with 0,1,2,3 denoting */
/* H,A,D,T and 4 denoting an erased bar. */

void decode_the_bars(void)
{
int rec[64]; /*received vector (over GF(64)) accepted by decoder */
int cor[64]; /* corrected vector (over GF(64)) produced by decoder */
int a[11], b[11], u[11], v[11]; /*the polynomials in the recursive sequences */
int deg_a, deg_b, deg_v; /*the degrees of these polynomials */
int syn[10]; /*syndrome vector */
int dv[11]; /* the derivative of polynomial v */
int delta, zeta, tau, r, s, t, w, temp; /*temporary scratchpad variables */
int i, j, k; /* loop counters */
int errcount, alpha_i; /*counts the errors, holds alpha^i during correction */
int erase_num; /* counts the number of erased symbols received */
int eraseloc[10]; /* holds locations (0...n) of the erasures */
int bound; /* bound for iterations of Euclidean Algorithm */
int bad_char_check;/* checks for bad characters after decoding */

/* Unpack the bars into a received vector rec[] */
for(j=2,k=15; k>=0;j+=3,k--)
        {if ((bars[j] == 4) || (bars[j+1]==4) || (bars[j+2]==4)) rec[k] = 64;
         else rec[k] = (bars[j]<<4) ^ (bars[j+1]<<2) ^ bars[j+2];}

/*=================================================== */
/*                    The Error Correction Routine */
/* */
/*  ========  Initialize, Catalogue the Erasures ========== */
/*                    and Calulate the Syndrome */

erase_num = 0;
decode_flag = 1;

for(j=0; j<n; j++)        /* Count the erasures and set the erased symbols to 0. *
        if (rec[j] >= 64)  /* Erased symbols come in coded as 64. */
                {eraseloc[erase_num] = j;
                erase_num++;
                rec[j] = 0;
                }
for(i=0; i<n; i++) cor[i] = rec[i]; /*copy the received rec[] into corrected cor

if(erase_num > erase_max)        /* Check for too many erasures */
        { decode_flag = 2;
        post_code[0] = Aletter[cor[14]];
        post_code[1] = Ndigit[cor[13]>>2];
        post_code[2] = Aletter[((cor[13]&3)<<4)^(cor[12]>>2)];
        post_code[3] = Ndigit[((cor[12]&3)<<2)^(cor[11]>>4)];
        post_code[4] = Aletter[((cor[11]&15)<<2)^(cor[10]>>4)];
        post_code[5] = Ndigit[cor[10]&15];
        DCI = Zchar[cor[15]];
        for(i=0;i<4;i++) mach_id[i] = bars[i+50];
        return;}

temp = 0;
for(k=0;k<m;k++)                /* Calculate the syndrome */
        {
        w = alpha[k]; s = rec[n-1];
        for (j = n-2;j >=0;j--) s = rec[j] ^ prod[s][w];
        syn[k] = s;
        temp = temp || s;  /* temp is checking for a non-zero syndrome */
        }
if (temp == 0)   /* if the syndrome is 0 exit */
        { decode_flag = 0;
        post_code[0] = Aletter[cor[14]];
        post_code[1] = Ndigit[cor[13]>>2];
        post_code[2] = Aletter[((cor[13]&3)<<4)^(cor[12]>>2)];
        post_code[3] = Ndigit[((cor[12]&3)<<2)^(cor[11]>>4)];
        post_code[4] = Aletter[((cor[11]&15)<<2)^(cor[10]>>4)];
        post_code[5] = Ndigit[cor[10]&15];
        DCI = Zchar[cor[15]];
        for(i=0;i<4;i++) mach_id[i] = bars[i+50];
        return;}

/*  =======  Setup for Euclidean Algorithm ========== */

for(k = 0;k<=m;k++)   /* initial values for poly. in Euclidean Alg. */
        { a[k] = u[k] = v[k] = 0;
        b[k] = syn[k];
        }
a[m] = 1; v[0] = 1; deg_a = m; deg_v = 0;

if(erase_num > 0)      /* initialization for erasures */
        { for(k = 0 ; k < erase_num ; k++)
                { deg_v++;
                w = alog[eraseloc[k]];
                for(j=deg_v ; j>0  ; j--) v[j] = v[j] ^ prod[w][v[j-1]];
                }
        for(k=0; k<m; k++)
                for (i=1; i<=k; i++)
                        b[k] = b[k] ^ prod[v[i]][syn[k-i]];
        }
for(deg_b=m-1; (b[deg_b]==0)&&(deg_b>0); deg_b--);  /* Calc. degree of b[] */
bound = (m/2) + (erase_num >>1);     /*Calc. the bound for iterations below */

/*  =======  Euclidean Algorithm ========== */

while (deg_b >= bound)
        {
        delta = deg_a - deg_b;
        w = inv[b[deg_b]];
        for(zeta=0; zeta <=delta; zeta++)
                {
                tau = prod[a[deg_a - zeta]][w];
                for(k=0; k <= deg_b; k++)
                        {
                        a[delta - zeta + k] ^= (prod[b[k]][tau]);
                        u[delta - zeta + k] ^= (prod[v[k]][tau]);
                        }
                }
        for(k = 0; k <=deg_a; k++)
                {temp = a[k]; a[k] = b[k]; b[k] = temp;}
        for(k = 0; k <= m; k++)
                {temp = u[k]; u[k] = v[k]; v[k] = temp;}
        deg_a = deg_b;
        for(deg_b = deg_b - 1; (b[deg_b] == 0)&&(deg_b>0); deg_b--);
    deg_v += delta;
        }

 /* b[] is now the error evaluator; v[] is the error locator */

 /*============ Make the Corrections ============ */

 for(i=0; i < deg_v; i +=2)  /* Calc. the derivative of the error locator */
        {dv[i] = v[i+1]; dv[i+1] = 0;}

 errcount = 0; /*errcount will record the number of errors found */
 alpha_i = 1;
 for(i=0;i<n;i++)
        { w = inv[alpha_i];
        s=v[deg_v]; for(k=deg_v-1; k >=0; k--) s = prod[s][w]^v[k];
        if (s == 0)
                { errcount++;
        r=dv[deg_v-1]; for(k=deg_v-2; k >=0; k--) r = prod[r][w]^dv[k];
                t=b[deg_b]; for(k=deg_b-1; k>=0; k--) t = prod[t][w]^b[k];
                cor[i] = cor[i] ^ prod[t][inv[r]];
        }
        alpha_i = by_alpha(alpha_i);
        }

 if ( errcount < deg_v)
        { decode_flag = 3;
        for(j=0; j<n; j++) cor[j] = rec[j];}

 post_code[0] = Aletter[cor[14]];
 post_code[1] = Ndigit[cor[13]>>2];
 post_code[2] = Aletter[((cor[13]&3)<<4)^(cor[12]>>2)];
 post_code[3] = Ndigit[((cor[12]&3)<<2)^(cor[11]>>4)];
 post_code[4] = Aletter[((cor[11]&15)<<2)^(cor[10]>>4)];
 post_code[5] = Ndigit[cor[10]&15];
 DCI = Zchar[cor[15]];
 for(i=0;i<4;i++) mach_id[i] = bars[i+50];
 /* and finally, check for any unacceptable characters */
 bad_char_check = (DCI == bad_char);
 for(i=0;i<6;i++) bad_char_check |= (post_code[i] == bad_char);
 if (bad_char_check ==1) decode_flag = 4;
 return;
 }
 /*=============================================================== */
 /*                    Print the decoded data */
 /* =============================================================== */
 void print_data(void)
 {
 int j; /* loop counter and temporary variable */
 char display[5]; /* table converting 0,1,2,3,4 to B,U,D,T,e. */

 display[0] = (char) 72; display[1] = (char) 65;
 display[2] = (char) 68; display[3] = (char) 84;
 display[4] = (char) 101; /* erasurers print as 'e' */
```

```
switch(decode_flag)
        {case 0: break;
        case 1: break;
        case 2: printf("Failed - too many erasures.\n"); break;
        case 3: printf("Failed - too many errors.\n"); break;
        case 4: printf("Decoding failure.\n"); break;
        default: {printf("Unknown decode flag value received.\n");}
        }

printf("DCI: %c",DCI);
printf("  Postal code:  ");
for(j=0; j<3;j++) printf("%c",(char) post_code[j]);
printf(" ");
for(j=3; j<6;j++) printf("%c",(char) post_code[j]);

printf("  Machine ID: "); for(j=0; j<4;j++) printf("%c",(char) display[mach_id[j
printf("\n");
return;
}
```

```
Program Listing - 94-10-12
Decoder for Postbar.D07, Postbar.D12 and Postbar.D22.


#include <stdio.h>
#include <math.h>

/*Constants */
const int m = 4; /* number of check symbols */
const int erase_max = 4; /* max. number of erasure corrections allowed */
const int bad_char = 35; /* code for bad characters (print as "#") */

/*Global Variables */

/*Static Variables: defined by setup(); used by decode_the_bars() */
int prod[64] [64], inv[64], alog[64]; /*mult., inv. and anti-log. tables */
int Atable[26];  /* the 6 bit (field representation) of letters using A table */
int Aletter[64]; /* the inverse of the table Atable[] */
int Ntable[10];  /* the 4 bit representations of the digits using N table */
int Ndigit[16];  /* the inverse of Ntable[] */
int Ztable[37]; /* The 6 bit representations of alphanumerics using Z table */
int Zchar[64]; /* inverse of the Ztable[] */
int alpha[4]; /* defining roots for the error correcting code */

/*Dynamic Variables: input to decode_the_bars() */
int bars[79]; /* holds the bar code pattern in numbers */
int code_id; /* holds code id. used by the decoder */
             /* (from DCI or number of bars) */
             /* code_id = 0 for D07, = 1 for D12, = 2 for D22 */

/*Dynamic Variables: output from decode_the_bars */
int DCI; /* holds the DCI in ascii after decoding */
int data_field[20];/* holds the AL and Cust. fields in ascii after decoding */
int decode_flag; /* flag indicating the result of the decoding */
int post_code[6]; /* ascii for the 6 chars. of the postal code after decoding */

/* Function Prototypes */

/* principal routines */
void setup(void); /* builds the tables; */
void get_decoder_input(char *); /* gets input for decode_the_bars */
void decode_the_bars(void); /* decodes the bars into postal data */
void print_data(void); /* prints the output data */

/* utilities used by setup() */
int by_alpha(int u); /* utility that multiplies u by alpha in the finite field *
int shift(int u); /* utility that shifts ascii values for use in Atable, Ntable
int zshift(int u); /* utility that shifts ascii values for use in Ztable */

/*Main Program */

int main(argc,argv)

int argc;
```

```
 char *argv[];

 {
 FILE *fd;
 char *cp;
 char buf[1024];
 int i,nline;

 if (argc < 2) {
    printf("Decoder program for the D codes (batch version).\n\n");
    printf("Usage: bpdecd filename\n\n");
    printf("  filename contains barcodes to be decoded.\n");
    exit();
 }

 cp = argv[1];

 fd = fopen(cp,"r");
 if (!fd) {
    printf("Unable to open file %s.\n",cp);
    exit();
 }

 setup();

 nline = 0;
 while (fgets(buf,sizeof(buf),fd)) {
    ++nline;
    if (buf[strlen(buf)-1] == '\n') buf[strlen(buf)-1] = '\0';
    for (i=0; i<strlen(buf); i++) {
       if (buf[i] < '0' || buf[i] > '4') {
          fprintf(stderr,"ERROR - Invalid input at line %d.\n",nline);
          fprintf(stderr,"        Invalid character at position %d.\n",i);
          exit();
       }
    }
    get_decoder_input(buf);
    decode_the_bars();
    print_data();
 }
 }

/*=================================================================== */
/*                        Initialization Routines */
/*=================================================================== */

int by_alpha(int u)
{
u = (u << 1);
if (u > 63) u =u^67;
return(u);
}

int shift(int u) /* changes ascii to indices for A and N tables */
{ if(u>64) u-=65;
else u -= 48;
return(u);
}
```

```
int zshift(int u) /* changes ascii to indices for Z table */
{ if(u>64) u-=65;
else if (u==32) u=36;
else u -= 22;
return(u);
}

void setup(void) /* The look-up tables are created */
{
int w0, w1;
int i, j, k, u, v;

/* Initialize the look-up symbology tables  */
/* The tables Atable[], Ntable[], Ztable[] convert a shifted ascii */
/* argument to the representations for the A,N and Z fields respectively. */
/* Shfted ascii means (ascii -65) for A .. Z and (ascii - 48) for 0..9 */

Atable[0] = 2; Atable[1] = 4; Atable[2] = 5; Atable[3] = 6;
Atable[4] = 8; Atable[5] = 9; Atable[6] = 10; Atable[7] = 1;
Atable[8] = 17; Atable[9] = 18; Atable[10] = 20; Atable[11] = 21;
Atable[12] = 0; Atable[13] = 24; Atable[14] = 25; Atable[15] = 26;
Atable[16] = 32; Atable[17] = 33; Atable[18] = 34; Atable[19] = 36;
Atable[20] = 37; Atable[21] = 38; Atable[22] = 40; Atable[23] = 41;
Atable[24] = 42; Atable[25] = 16;

for(k=0; k<64; k++) Aletter[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Aletter[Atable[k]] = k+65;

Ntable[0] = 0; Ntable[1] = 1; Ntable[2] = 2; Ntable[3] = 4; Ntable[4] = 5;
Ntable[5] = 6; Ntable[6] = 8; Ntable[7] = 9; Ntable[8] = 10; Ntable[9] = 12;

for(k=0; k<16; k++) Ndigit[k] = bad_char; /* unsuitable characters print as # */
for(k=0; k<10;k++) Ndigit[Ntable[k]] = k+48;

Ztable[0] = 0; Ztable[1] = 1; Ztable[2] = 2; Ztable[3] = 4; Ztable[4] = 5;
Ztable[5] = 6; Ztable[6] = 8; Ztable[7] = 9; Ztable[8] = 10; Ztable[9] = 16;
Ztable[10] = 17;Ztable[11] = 18;Ztable[12] = 20;Ztable[13] = 21;Ztable[14] = 22;
Ztable[15] = 24;Ztable[16] = 25;Ztable[17] = 26;Ztable[18] = 32;Ztable[19] = 33;
Ztable[20] = 34;Ztable[21] = 36;Ztable[22] = 37;Ztable[23] = 38;Ztable[24] = 40;
Ztable[25] = 41;Ztable[26] = 42;Ztable[27] = 48;Ztable[28] = 49;Ztable[29] = 50;
Ztable[30] = 52;Ztable[31] = 53;Ztable[32] = 54;Ztable[33] = 56;Ztable[34] = 57;
Ztable[35] = 58;Ztable[36] = 3;

for(k=0; k < 64; k++) Zchar[k] = bad_char; /* unsuitable characters print as # *
for(k=0; k<26; k++) Zchar[Ztable[k]] = k + 65;  /* ascii for the letters */
for(k=26; k<36; k++) Zchar[Ztable[k]] = k + 22; /* ascii for the digits */
Zchar[3] = 32; /* ascii for a space */

/* the prod[][] and inv[] tables are defined */
for(k = 0; k < 64;k++) {prod[0][k] = 0; prod[1][k] = k;}

w0 = 1; alog[0] = 1;
for(u=1;u<64;u++)
        {w1 = by_alpha(w0);
        alog[u] = w1;
        for(v=0;v<64;v++)
                {
                prod[w1][v] = by_alpha(prod[w0][v]);
                if (prod[w1][v] == 1) inv[w1] = v;
                }
        w0 = w1;)

/* These are the codes for the roots of the generator poly. */
/* They are alpha^1, ... , alpha^10. */
for (k=0;k<m;k++) alpha[k] = alog[k+1];

return;
}
/*================================================== */
/*                    Makes Test Input for Decoder */
/*================================================== */
void get_decoder_input(char *buf)
{
int i;

for (i=0; i<strlen(buf); i++) bars[i] = buf[i] - '0';
if (strlen(buf) > 49)
   code_id = 2;
else if (strlen(buf) > 34)
   code_id = 1;
else
   code_id = 0;

return;
}
/*============================================================= */
/*                        Decode the Bars  */
/*============================================================= */
/* The main decoding routine. */
/* The bar pattern is received in bars[] with 0,1,2,3 denoting */
/* H,A,D,T and 4 denoting an erased bar. */
/* The barcode used is identified by code_id = 0 for D07, */
/*       = 1 for D12, = 2 for D22. */

void decode_the_bars(void)
{
int i,j,k; /* loop counters */
int n,N; /* code length in characters(n) and in bars(N) */
int a[5], b[5], u[5], v[5]; /*the polys in the recursive sequences */
int rec[26]; /*received vector accepted by decoder */
int cor[26]; /* corrected vector produced by decoder */
int deg_a, deg_b, deg_v; /*the degrees of these polynomials */
int syn[4]; /*syndrome vector */
int dv[5]; /* the derivative of polynomial v */
int delta, zeta, tau, r, s, t, w, temp; /*temporary scratchpad variables */
int errcount, alpha_i; /*counts the errors, holds alpha^i during correction */
int erase_num; /* counts the number of erased symbols received */
int eraseloc[5]; /* holds locations (0...n) of the erasures */
int bound; /* bound for iterations of Euclidean Algorithm */
int bad_char_check;/* flag for bad characters after decoding */

/*calculate code length in symbols(n) and bars(N) */
switch(code_id)
        {case 0: {n=10;break;}
        case 1: {n=15;break;}
        case 2: n=25;}

/* Unpack the bars into a received vector rec[] */
for(j=2,k=n-1;k>=0;j+=3,k--)
        if((bars[j]==4)||(bars[j+1]==4)||(bars[j+2]==4)) rec[k]=64;
```

```
          else rec[k] = (bars[j]<<4)^(bars[j+1]<<2)^bars[j+2];

/*  =======  Initialize, Catalogue the Erasures ==========  */
/*                  and Calulate the Syndrome */
erase_num = 0;
decode_flag = 1;

for(j=0; j<n; j++)        /* Count the erasures and set the erased symbols to 0. *
        if (rec[j] >= 64)  /* Erased symbols come in coded as 64. */
                {eraseloc[erase_num] = j;
                erase_num++;
                rec[j] = 0;
                }
for(i=0; i<n; i++) cor[i] = rec[i]; /*copy the received rec[] into corrected cor

if(erase_num > erase_max)         /* Check for too many erasures */
        { decode_flag = 2; return;}

temp = 0;
for(k=0;k<m;k++)                  /* Calculate the syndrome */
        {
        w = alpha[k]; s = rec[n-1];
        for (j = n-2;j >=0;j--)  s = rec[j] ^ prod[s][w];
        syn[k] = s;
        temp = temp || s;  /* temp is checking for a non-zero syndrome */
        }
if (temp == 0)    /* if the syndrome is 0 exit */
        { decode_flag = 0;
        DCI = Zchar[cor[n-1]];
        if (code_id == 2) for(i=0;i<15;i++) data_field[i] = Zchar[cor[n-7-i]];
        if (code_id == 1) for(i=0;i<5;i++) data_field[i] = Zchar[cor[n-7-i]];
        post_code[0] = Aletter[cor[n-2]];
        post_code[1] = Ndigit[cor[n-3]>>2];
        post_code[2] = Aletter[((cor[n-3]&3)<<4)^(cor[n-4]>>2)];
        post_code[3] = Ndigit[((cor[n-4]&3)<<2)^(cor[n-5]>>4)];
        post_code[4] = Aletter[((cor[n-5]&15)<<2)^(cor[n-6]>>4)];
        post_code[5] = Ndigit[cor[n-6]&15];
        return;}

/*  =======  Setup for Euclidean Algorithm ==========  */

for(k = 0;k<=m;k++)    /* initial values for poly. in Euclidean Alg. */
        { a[k] = u[k] = v[k] = 0;
        b[k] = syn[k];
        }
a[m] = 1; v[0] = 1; deg_a = m; deg_v = 0;

if(erase_num > 0)      /* initialization for erasures */
        { for(k = 0 ; k < erase_num ; k++)
                { deg_v++;
                w = alog[eraseloc[k]];
                for(j=deg_v ; j>0  ; j--) v[j] = v[j] ^ prod[w][v[j-1]];
                }
        for(k=0; k<m; k++)
                for (i=1; i<=k; i++)
                        b[k] = b[k] ^ prod[v[i]][syn[k-i]];
        }
for(deg_b=m-1; (b[deg_b]==0)&&(deg_b>0); deg_b--);  /* Calc. degree of b[] */
bound = (m/2) + (erase_num >>1);     /*Calc. the bound for iterations below */
```

```
 /*  =======  Euclidean Algorithm ========== */

 while (deg_b >= bound)
         {
         delta = deg_a - deg_b;
         w = inv[b[deg_b]];
         for(zeta=0; zeta <=delta; zeta++)
                 {
                 tau = prod[a[deg_a - zeta]][w];
                 for(k=0; k <= deg_b; k++)
                         {
                         a[delta - zeta + k] ^= (prod[b[k]][tau]);
                         u[delta - zeta + k] ^= (prod[v[k]][tau]);
                         }
                 }
         for(k = 0; k <=deg_a; k++)
                 {temp = a[k]; a[k] = b[k]; b[k] = temp;}
         for(k = 0; k <= m; k++)
                 {temp = u[k]; u[k] = v[k]; v[k] = temp;}
         deg_a = deg_b;
         for(deg_b = deg_b - 1; (b[deg_b] == 0)&&(deg_b>0); deg_b--);
     deg_v += delta;
         }

/* b[] is now the error evaluator; v[] is the error locator */

/*============ Make the Corrections ========== */

for(i=0; i < deg_v; i +=2)  /* Calc. the derivative of the error locator */
        {dv[i] = v[i+1]; dv[i+1] = 0;}

errcount = 0; /*errcount will record the number of errors found */
alpha_i = 1;
for(i=0;i<n;i++)
        { w = inv[alpha_i];
        s=v[deg_v]; for(k=deg_v-1; k >=0; k--) s = prod[s][w]^v[k];
        if (s == 0)
                { errcount++;
        r=dv[deg_v-1]; for(k=deg_v-2; k >=0; k--) r = prod[r][w]^dv[k];
                t=b[deg_b]; for(k=deg_b-1; k>=0; k--) t = prod[t][w]^b[k];
                cor[i] = cor[i] ^ prod[t][inv[r]];
        }
        alpha_i = by_alpha(alpha_i);
        }

if ( errcount < deg_v)
        { decode_flag = 3;
        for(j=0; j<n; j++) cor[j] = rec[j];}

/*define the output */

DCI = Zchar[cor[n-1]];
if (code_id == 2) for(i=0;i<15;i++) data_field[i] = Zchar[cor[n-7-i]];
if (code_id == 1) for(i=0;i<5;i++) data_field[i] = Zchar[cor[n-7-i]];
post_code[0] = Aletter[cor[n-2]];
post_code[1] = Ndigit[cor[n-3]>>2];
post_code[2] = Aletter[((cor[n-3]&3)<<4)^(cor[n-4]>>2)];
post_code[3] = Ndigit[((cor[n-4]&3)<<2)^(cor[n-5]>>4)];
post_code[4] = Aletter[((cor[n-5]&15)<<2)^(cor[n-6]>>4)];
post_code[5] = Ndigit[cor[n-6]&15];
```

We claim:

1. A mail piece bearing a bar codeword containing information for the processing of the mail piece, the bar codeword having a plurality of parallel bars each of which has a state selected from a plurality of possible states, the bar codeword comprising a start field followed by a data content identifier (DCI) field specifying the structure of the codeword followed by at least one data field, followed by a Reed-Solomon parity field, followed by a stop field.

2. A mail piece according to claim 1 in which the start field and the stop field are identical, each consisting of two different state bars.

3. A mail piece according to claim 1 in which the Reed-Solomon field contains a plurality of Reed-Solomon characters each encoded in 3 bars.

4. A mail piece according to claim 1 in which the DCI is encoded in three bars.

5. A mail piece according to claim 1 in which the at least one data field includes at least one of a postal code, with or without address locator, customer information and service information.

6. A mail piece according to claim 5 in which the at least one data field contains characters each of which is encoded in three bars.

7. A mail piece according to claim 5 in which the postal code contains alphabetic characters and numeric characters, the alphabetic characters being encoded in three bars and the numeric characters being encoded in two bars, any remaining data being encoded in three bar characters.

8. A mail piece according to claim 1 in which the at least one data field includes a country code followed by at least one of a postal code, with or without an address locator, customer information and service information.

9. A mail piece according to claim 8 in which the at least one data field contains characters each of which is encoded in three bars.

10. A mail piece according to claim 1 in which the at least one data field includes a postal code and in which a further data field comprising machine ID follows the Reed-Solomon parity field.

11. A mail piece according to claim 10 in which the machine ID is represented by four bars and each bar state has a unique numerical value.

12. A mail piece according to claim 1 in which a bar code sequencer field is located between the DCI field and the at least one data field, the bar code sequencer field containing a code indicating one of the bar code is a single code, the bar code is the first of two concatenated bar codes and the bar code is the second of two concatenated bar codes.

13. A mail piece according to claim 1 in which the codeword may be applied by one of the Post Office and a customer, the Reed-Solomon parity field being longer for a Post Office applied codeword than for a customer applied codeword.

14. A mail piece bearing a bar codeword containing information for the processing of the mail piece, the bar codeword having a plurality of parallel bars each of which has one of four possible states selected from a full height bar

(H), extending between an upper and lower level, a partial height bar (D) descending to the lower level, a partial height bar (A) ascending to the upper level, and a bar (T) the height and position of which is determined by overlap of the descending and ascending bars, the bar codeword comprising a start field followed by a data content identifier (DCI) field specifying the structure of the codeword followed by at least one data field followed by a Reed-Solomon parity field followed by a stop field.

15. A mail piece according to claim 14 in which the start field and the stop field are identical, each consisting of two different state bars.

16. A mail piece according to claim 15 in which the start and stop bars are an A bar followed by a T bar.

17. A mail piece according to claim 14 in which the DCI is encoded in three bars.

18. A mail piece according to claim 14 in which the at least one data field includes at least one of a postal code, with or without address locator, customer information and service information.

19. A mail piece according to claim 15 in which the at least one data field contains characters each of which is encoded in three bars.

20. A mail piece according to claim 18 in which the postal code contains alphabetic characters and numeric characters, the alphabetic characters being encoded in three bars and the numeric characters being encoded in two bars, any remaining data being encoded in three bar characters.

21. A mail piece according to claim 14 in which the at least one data field includes a country code followed by at least one of a postal code, with or without an address locator, customer information and service information.

22. A mail piece according to claim 21 in which the at least one data field contains characters each of which is encoded in three bars.

23. A mail piece according to claim 14 in which the at least one data field includes a postal code and in which a further data field comprising machine ID follows the Reed-Solomon parity field.

24. A mail piece according to claim 23 in which the machine ID is represented by four bars and each bar state has a unique numerical value.

25. A mail piece according to claim 14 in which a bar code sequencer field is located between the DCI field and the at least one data field, the bar code sequencer field containing a code indicating one of the bar code is a single code, the bar code is the first of two concatenated bar codes and the bar code is the second of two concatenated bar codes.

26. A mail piece according to claim 14 in which the codeword may be applied by one of the Post Office and a customer, the Reed-Solomon parity field being longer for a Post Office applied codeword than for a customer applied codeword.

27. A mail piece according to claim 14 in which the Reed-Solomon field contain a plurality of Reed-Solomon characters each encoded in 3 bars.

* * * * *